ALGORITMOS

Y ESTRUCTURAS DE DATOS

APUNTE DE TEORIA

AUTORA: Ing. ESTELA M. SORRIBAS

ALGORITMOS Y ESTRUCTURAS DE DATOS

► INTRODUCCIÓN:

El desarrollo de la tecnología de la información y de las comunicaciones, ha sido responsable de una buena parte de los cambios sociales y productivos en el mundo de las últimas décadas.

Las sociedades se distinguen entre sí por la complejidad de los problemas que puedan resolver, para lo cual deben acceder al conocimiento. Este acceso al conocimiento depende de cómo se procesa, almacena y trasmite la información en un país. Para brindar respuesta a esta necesidad social, la educación juega un papel muy importante.

Una de las prioridades de los sistemas educativos de los países que pretendan un crecimiento económico y un desarrollo social sustentable, es la alfabetización en tecnología.

El área Programación tiene como objetivo formar e informar acerca de metodologías, técnicas y lenguajes de programación, como herramientas básicas para el desarrollo de software y el estudio de disciplinas que permitan crear nuevas tecnologías.

► OBJETIVOS:

Esta asignatura, tiene como primer objetivo presentar a la programación como el arte o la técnica de construir y formular algoritmos en forma sistemática. Se debe aprender a proceder metódica y sistemáticamente en el diseño de algoritmos mediante la demostración de problemas y técnicas que son típicas de la programación, pero independientes del área de aplicación en particular. Por esta razón, ningún área de aplicación específica se enfatiza como un fin. Con el mismo espíritu se resta importancia a la notación y al lenguaje de programación, el lenguaje es nuestra herramienta, no un fin en sí mismo.

El fin primario de los cursos de programación no debe ser enseñar la perfección en el conocimiento de todas las características e idiosincrasias de un lenguaje en particular, sino entender el problema que se nos pide resolver y resolverlo de una manera comprensible y sobre todo confiable.

● Objetivos Generales:

- Mejorar la capacidad de razonamiento.
- Adquirir habilidad y seguridad en la resolución de problemas.
- Desarrollar aptitudes que les permitan seguir aprendiendo por sí mismos.

- Desarrollar hábitos de observación, razonamiento, orden, autocrítica y trabajo metódico.
- Fomentar la participación activa creando un ambiente de aprendizaje creativo.

● Objetivos Específicos:

- Formular un problema en forma correcta, completa y sin ambigüedades.
- Utilizar los conocimientos adquiridos para elegir un método para hallar la solución de los problemas.
- Expresar el método elegido de forma tal que pueda ser interpretado por el procesador a utilizarse.
- Reconocer datos e incógnitas
- Elegir correctamente la estructura de datos.
- Ejecutar el procedimiento elegido para obtener la solución del problema.
- Expresar el algoritmo en lenguaje de programación.

► CONTENIDOS:

La secuencia de contenidos conceptuales se organizó en siete unidades didácticas:

◊ Unidad 1:

Algoritmo, Programa, Lenguaje de programación, Lenguaje de máquina,

Compilador – Definiciones

Representación de algoritmos – Diagramación – Diagramas de

Nassi- Schneiderman o de Chapin.

Diseño general de un algoritmo: Partes básicas.

Constantes y variables – Identificadores – Asignación – Operadores Matemáticos

Operadores relacionales – Definición.

◊ Unidad 2:

Introducción al Pascal – Programa Pascal – Encabezamiento – Bloque – Cuerpo

Declaraciones y Definiciones – Tipos de Datos standard: enteros, reales,

caracteres y lógicos

Cuerpo – Sentencia de asignación – Sentencia de entrada – Sentencia de Salida – Salidas formateadas

◊ Unidad 3:

Estructuras de Control
Bifurcación ó Selección Simple – Diagrama y programa -
Repetición ó Iteración: con cantidad conocida de veces y con cantidad desconocida de veces – Diagrama y programa
Selección múltiple – Diagrama y programa

◊ Unidad 4:

Tipos de Datos no standard o definidos por el programador
Tipo Enumerado ó Escalar
Tipo Subrango ó Intervalo
Tipo Estructurado – Arreglos unidimensionales y multidimensionales
Ordenamiento de un arreglo unidimensional – Búsqueda de un valor en un arreglo unidimensional: Búsqueda Secuencial y Búsqueda Dicotómica
Intercalación de arreglos unidimensionales ordenados
Ordenamiento y Búsqueda en un arreglo bidimensional

◊ Unidad 5:

Subprogramas – Definición
Funciones y Procedimientos – Definiciones – Diferencias
Variables Locales – Variables Globales
Correspondencia Argumento-Parámetro
Parámetro por valor – Parámetro por Referencia ó por Variable

◊ Unidad 6:

Otras Estructuras de Datos
Registros – Registros Jerárquicos
Arreglos de Registros

◊ Unidad 7:

Archivos – Introducción - Buffers

Operaciones básicas sobre archivos – Otras operaciones sobre archivos

Organización y acceso a un archivo

► BIBLIOGRAFIA:

- ***INTRODUCCIÓN A LA PROGRAMACIÓN Y A LAS ESTRUCTURAS DE DATOS***
  Silvia Braunstein ; Alicia Gioia - EUDEBA
- ***INTRODUCCIÓN AL PASCAL***
  Nell Dale ; Orshalick – McGraw Hill
- ***PASCAL MAS ESTRUCTURAS DE DATOS***
  Nell Dale ; Susan Lilly – McGraw Hill
- ***ALGORITMOS, DATOS Y PROGRAMAS CON APLICACIONES EN PASCAL, DELPHI Y VISUAL DA VINCI***
  Armando E. de Giusti – Prentice Hall
- ***ALGORITMOS + ESRUCTURAS DE DATOS = PROGRAMAS***
  Niklaus Wirth – El Ateneo

UNIDAD Nº 1

1-1. INTRODUCCIÓN

Dentro de los objetivos planteados para esta asignatura aparecen palabras tales como: algoritmos, programas, lenguaje de programación, etc., con las cuales no estamos familiarizados; para entenderlos mejor veamos algunas definiciones:

1-1-1. ALGORITMOS:

Secuencia de acciones o pasos que permite resolver un problema. Un mismo problema puede ser resuelto con distintos algoritmos.-

1-1-2. PROGRAMA:

Traducción o codificación de un algoritmo a un lenguaje de programación; o sea, una secuencia de instrucciones que indica las acciones que han de ejecutarse.-

1-1-3. LENGUAJE DE PROGRAMACION:

Conjunto de reglas, símbolos y palabras especiales utilizadas para construir un programa.-

1-1-4. LENGUAJE DE MAQUINA:

Lenguaje usado directamente por la computadora y compuesto de instrucciones codificadas en sistema binario.-

1-1-5. COMPILADOR:

Programa que traduce un programa escrito en lenguaje de alto nivel a lenguaje de máquina.-

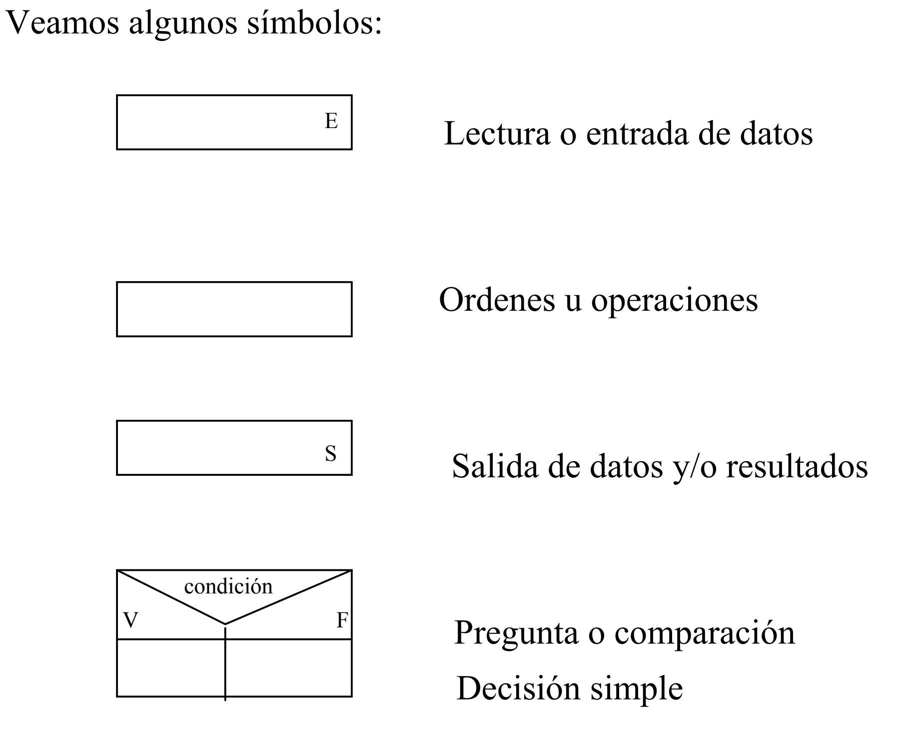
1-2. DIAGRAMACION:

Una vez comprendido el problema, se hace una representación gráfica de los pasos a seguir para resolverlo. Esta representación se llama diagramación.-

Los diagramas son un conjunto de símbolos que han convenido de distintas maneras distintos autores. En este curso adoptaremos los diagramas de NASSI-

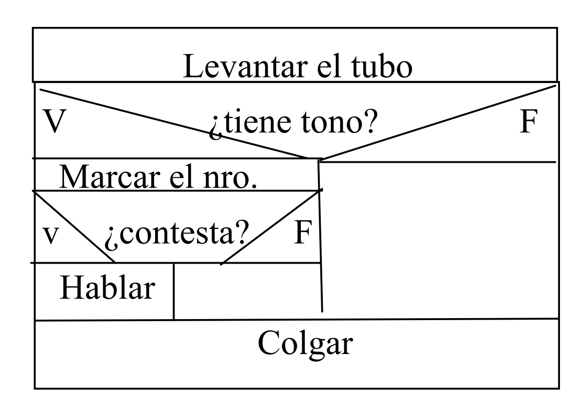
SCHNEIDERMAN, más conocidos como diagramas de CHAPIN; que permiten representar adecuadamente las estructuras de control de los lenguajes estructurados como PASCAL.-

Veamos algunos símbolos:



EJEMPLO:

Describir mediante un diagrama de Chapin, el procedimiento lógico que debe ser seguido para hablar por teléfono.-



1-3. <u>CONSTANTES Y VARIABLES:</u>

1-3-1. <u>IDENTIFICADOR:</u>

Como en el álgebra, a cada dato o elemento, ya sea constante o variable, se le bautiza con un nombre o identificador; el cual si se ha elegido adecuadamente, ayuda mucho a la persona que lea el programa.-

1-3-2. <u>CONSTANTES:</u>

Como su nombre lo indica, son datos que no varían durante la ejecución de un programa.-

1-3-3. <u>VARIABLES:</u>

Son datos que cambian o evolucionan durante la vida o ejecución de un programa.-

1-4. <u>SENTENCIA DE ASIGNACION:</u>

Asigna el valor de la expresión que está a la derecha del signo **:=** (signo de asignación en lenguaje Pascal), a la variable que está a la izquierda. En el diagrama este signo puede ser reemplazado por ←

Ejemplo: NUM := 6 + R (siendo conocido el valor de R)

1-5. <u>OPERADORES:</u>

+	Suma
-	Resta
*****	Multiplicación
/	División real
DIV	División entera
MOD	Resto de la división entera

1-6. OPERADORES RELACIONALES:

=	Igual
<>	Distinto
<	Menor
>	Mayor
<= ó =<	Menor o igual
>= ó =>	Mayor o igual
OR	ó lógico
AND	y lógico
NOT	no

1-7. EJEMPLO:

Realizar un algoritmo en diagrama de Chapin para resolver el siguiente problema:

Dados como datos: el precio del kilowatt-hora, la lectura actual y la lectura anterior de un medidor; calcular el importe que una persona deberá abonar a la E.P.E. Mostrar los datos y el resultado obtenido.-

PRECIO, LECANT, LECACT	E
CONSUMO := LECACT - LECANT	
IMPORTE := CONSUMO * PRECIO	
PRECIO, LECANT, LECACT, IMPORTE	S

UNIDAD Nº 2

INTRODUCCION AL PASCAL

2-1. PROGRAMA EN PASCAL:

Un programa en lenguaje PASCAL está dividido en dos partes:

a)- ***ENCABEZAMIENTO:*** Este se compone de la palabra reservada PROGRAM seguida de un identificador (nombre del programa definido por el programador), y una lista de parámetros encerrados entre paréntesis que son los nombres de los ficheros a través de los cuales el programa se comunica con y desde el medio exterior. Nosotros utilizaremos INPUT (o sea entrada de datos por teclado), y OUTPUT (o sea salida de datos y/o resultados por pantalla).-

b)- ***BLOQUE:*** Este consta de dos partes:

i) DECLARACIONES Y DEFINICIONES:

- Declaración de etiquetas.

- Definición de constantes.

- Definición de tipos.

- Declaración de variables.

- Declaración de procedimientos y funciones.

Todas o algunas de ellas pueden no estar. No nos ocuparemos en este curso de la declaración de etiquetas. En este capítulo veremos definición de constantes y declaración de variables; y más adelante veremos las restantes.-

ii) CUERPO: Está compuesto por las sentencias ejecutables del programa encerradas entre BEGIN y END. .-

Los comentarios del programa deben ir entre (* *) o entre { }

2-2. TIPOS DE DATOS STANDARD:

2-2-1. INTEGER:

Son todos los datos que contienen números positivos o negativos sin parte decimal (o sea números enteros). Ellos constan de un signo y dígitos sin comas:

Ejemplos: -109 0 12 +35 -1

Cuando se omite el signo se asume que el número es positivo.-

Teóricamente, no hay límite sobre el tamaño de los enteros, pero las limitaciones del hardware de la computadora y las consideraciones prácticas dan lugar a que haya que limitar el tamaño de un entero. Puesto que este límite varía de unas máquinas a otras, Pascal tiene un identificador predefinido, MAXINT, cuyo valor es el mayor valor entero que puede representarse en la computadora; en general MAXINT es 32767 (aunque puede ser diferente para su máquina, por lo que puede imprimirlo para ver cuál es su valor), entonces el rango de los enteros permitido sería:

desde: -MAXINT hasta: MAXINT

ó

desde: -32767 hasta: 32767

2-2-2. REAL:

Son todos los datos que contienen números positivos o negativos que tienen una parte entera y una parte decimal.-

Cuando en un programa, a un dato se le asigne o se le introduzca un valor real, siempre debe usarse un punto decimal con al menos un dígito a cada lado; y si usamos la notación científica, o sea potencias de 10, debe colocarse una "E" antes del exponente.-

Ejemplos:

Reales válidos	Reales no válidos
12.34	12. (ningún dígito después del ".")
63E4	12.E3 (ningún dígito después del ".")
34.2E5	.46E-6 (ningún dígito antes del ".")
100E-9	245 (ni "E" ni ".")
-50E-4	56.5E (ningún dígito después de "E")

2-2-3. CHAR:

Son todos los datos que contienen un carácter alfanumérico (sólo uno). Los caracteres alfanuméricos incluyen dígitos, letras y símbolos especiales.-

Ejemplos: 'A' ; 'a' ; '4' ; '+' ; '?' ; '$' ; ' '

El compilador Pascal necesita las comillas simples o apóstrofos para diferenciar entre el dato carácter '4' ó '+' del entero 4 o el signo suma. Observe que el blanco (' ') es un carácter.-

No se puede sumar '4' y '9', pero puede comparar valores de tipo CHAR. El conjunto de caracteres de una máquina está ordenado, 'A' es siempre menor que 'B', 'B' es menor que 'C', y así sucesivamente. También '1' es menor que '2', '3' es menor que '4',etc.-

2-2-4. BOOLEAN:

Son los datos que contienen alguno de los dos valores:

TRUE o FALSE (verdadero o falso).-

Los datos Booleanos no pueden leerse como datos, pero pueden imprimirse.-

Cuando se aplican operadores lógicos (and($\wedge$); or($\vee$); y not($\neg$)) a operandos Booleans, producen un valor Boolean, que representaremos en el siguiente cuadro, siendo p y q dos operandos Boolean:

p	q	$p \vee q$	$p \wedge q$	$\neg p$
.T.	.T.	.T.	.T.	.F.
.T.	.F.	.T.	.F.	.F.
.F.	.T.	.T.	.F.	.T.
.F.	.F.	.F.	.F.	.T.

2-2-5. STRING:

Turbo Pascal proporciona el tipo string para el procesamiento de cadenas (secuencias de caracteres).

La definición de un tipo string debe especificar el número máximo de caracteres que puede contener, esto es, la máxima longitud para las cadenas de ese tipo. La longitud se especifica por una constante entera en el rango de 1 a 255.

El formato para definir un tipo string es : <identificador> = string [*limite_superior*];

Ejemplo:

```
Var
  nombre    : string[30];
  domicilio : string[30];
  ciudad    : string[40];
```

Una Vez declaradas las variables se pueden realizar asignaciones:

```
nombre    := 'Juan José Perez' ;
domicilio := 'Buenos Aires 2416 – 1ºpiso';
ciudad    := 'Rosario';
```

U operaciones de lectura/escritura:

```
ReadLn (nombre);
WriteLn('Hola ',nombre);
```

Es posible acceder a posiciones individuales dentro de una variable cadena, mediante la utilización de corchetes que dentro de ellos se especifíca el número indice dentro de la cadena a utilizar así para el ejemplo anterior se tiene :

nombre[1] ==> 'J'
nombre[2] ==> 'u'
nombre[3] ==> 'a'
nombre[4] ==> 'n'

Operaciones entre cadenas

Las operciones básicas entre cadenas son : *asignación, comparación y concatenación.* Es posible *asignar* una cadena a otra cadena, incluso aunque sea de longitud física más pequeña en cuyo caso ocurriría un truncamiento de la cadena.

Si la cantidad de caracteres asignados a una variable string es menor que la definición realizada, Pascal completa con blancos a la derecha de la cadena; si se le asigna una cantidad mayor, trunca

Ejemplo:

```
Var
  Nombre1 : String[9];
  Nombre2 : String[20];
   .
   .
   .
  Nombre1 := 'Instituto Tecnológico';
  Nombre2 := 'Universidad';
```

El resultado de la asignación en la variable Nombre1 será la cadena 'Instituto' y en la variable Nombre2 será ´Universidad ´

2-3. SENTENCIAS DE ENTRADA Y SALIDA:

En un equipo de cálculo, la entrada y salida representa la interfase final entre el usuario de un programa y el programa que ejecuta la computadora.-

Las sentencias de entrada de un programa hacen que la computadora lea los datos durante la ejecución del programa, lo que permite al programador escribir programas generales que pueden ser utilizados repetidamente para conjuntos de datos diferentes, cuyos valores no tienen porqué ser conocidos exactamente por el programador en el momento de escribir el programa.-

Sin las sentencias de salida, muchos programas serían inútiles ya que el usuario no tendría ninguna forma de conocer el resultado obtenido.-

2-3-1. SENTENCIA READ:

Es la sentencia de entrada en Pascal y permite la lectura de uno o varios datos y su asignación a una o varias variables del programa.-

Ejemplo:

```
READ (NUM);
```

Significa que le asigna a la variable NUM el valor que ingresamos como dato.-

Si queremos ingresar más de un dato, cada variable va separada de otra por coma, y los datos respectivos deben ir separados por uno o más blancos.-

Ejemplo:

```
READ (PRECIO, CANTIDAD);
```

2-3-2. SENTENCIA WRITE:

Es la sentencia de salida en Pascal y permite que el contenido de una variable o el valor de una expresión sea conocida por nosotros mediante una impresión.-

Ejemplo:

```
WRITE (IMPORTE);
```

Si la sentencia de lectura anterior representa el ingreso del precio de un determinado artículo y la cantidad vendida del mismo y deseáramos conocer cuál es el importe que debe pagar el cliente, el segmento de programa sería:

```
READ (PRECIO, CANTIDAD);
IMPORTE := PRECIO * CANTIDAD;
WRITE (IMPORTE);
```

Si quisiéramos imprimir también los datos leídos, la sentencia write quedaría:

```
WRITE (PRECIO, CANTIDAD, IMPORTE);
```

Pero dijimos que la sentencia write también permite conocer el valor de una expresión; así este mismo ejemplo podría haber sido resuelto de la siguiente manera:

```
READ (PRECIO, CANTIDAD);
WRITE (PRECIO * CANTIDAD);
```

Si deseáramos que los valores numéricos impresos estuvieran acompañados por un texto, es necesario que dicho texto aparezca en la sentencia write encerrado entre apóstrofos.-

Ejemplo:

```
WRITE ('EL IMPORTE DE LA FACTURA ES: ', IMPORTE);
```

Si quisiéramos que los datos se exhiban en un renglón y el resultado en otro, necesitaríamos utilizar la sentencia WRITELN. Esta sentencia provoca lo siguiente: una vez que la computadora exhibió lo indicado en la lista de parámetros encerrados entre paréntesis el cursor queda al principio del renglón siguiente.-

Ejemplo:

```
WRITELN (PRECIO, CANTIDAD);
WRITE (IMPORTE);
```

en este caso exhibiría los contenidos de las variables precio y cantidad en un renglón, y el de la variable importe en otro.-

Si utilizamos una sentencia WRITELN sin parámetros, obtenemos una línea en blanco.-

Algo similar ocurre con la sentencia READLN. Después de una sentencia READLN, la parte no leída de la línea actual de entrada de datos es saltada y la siguiente entrada será tomada de la siguiente línea.-

2-3-3. SALIDAS FORMATEADAS:

Se puede controlar la apariencia de una salida indicando cuántas columnas se quiere

que ocupe una variable, una constante o un mensaje.-

Si el contenido de una variable o constante es un valor entero o carácter, se pone un ':' seguido de un valor entero después de la variable o constante en la lista de parámetros de la sentencia WRITE o WRITELN. El entero que sigue a ':' indica cuántas columnas o posiciones ocupará sobre la línea, la variable o constante que se va a imprimir a partir del lugar donde está el cursor en ese momento.-

El valor de la variable o constante se imprimirá justificada a la derecha con blancos a la izquierda para ocupar el número correcto de columnas indicado después del ':'.-

EJEMPLO:

Si NUM = 25 (entero); CONT = 54 (entero) y LET = 'B'(carácter)

Sentencia	Salida ('-' representa blanco)
WRITE (NUM:4, CONT:5, LET:3)	--25---54--B
WRITE (NUM:3, CONT:2, LET:1)	-2554B
WRITE (NUM:6, LET:1)	----25B
WRITE (CONT:6, LET:4)	----54---B

El mismo criterio se aplica a los literales. Por ejemplo:

Sentencia	Salida
WRITE ('EL NUMERO ES ':16)	---EL-NUMERO-ES-
WRITE ('EL NUMERO ES ':15, NUM:3)	--EL-NUMERO-ES--25

Si queremos exhibir el valor de una variable o constante con contenido real y no le damos un formato, el valor se imprimirá en la notación E con un dígito antes del punto decimal, lo cual no es muy claro. Por lo tanto, si ese valor lo queremos imprimir en notación decimal, Pascal nos proporciona un formato para ello:

Si se coloca un ':' seguido de un valor entero y luego otro ':' seguido de otro valor entero; el primer número sigue especificando la cantidad total de columnas que se usarán (incluido el punto decimal), y el segundo número especifica la cantidad de

dígitos que se imprimirán después del punto decimal (si el número tiene más decimales que lo especificado en el formato, la computadora redondea el siguiente a la cantidad determinada).-

EJEMPLO: Si PROM = 23.346

Sentencia	Salida
WRITE (PROM:9:3);	---23.346
WRITE (PROM:10:2);	-----23.35
WRITE (PROM:10:4);	---23.3460
WRITE (PROM:6:0);	---23.

2-4. RESOLUCION DE UN PROBLEMA:

Dados como datos el precio del kilowatt-hora y las lecturas actual y anterior de un medidor, calcular el importe que una persona deberá abonar a la E.P.E.

Exhibir los datos en un renglón y el resultado en otro.-

2-4-1. DIAGRAMA DE CHAPIN:

PKW , LACT , LANT	E
CONS ← LACT - LANT	
IMPOR ← CONS * PKW	
PKW, LACT, LANT 'EL IMPORTE ES =', IMPOR	S

2-4-2. PROGRAMA EN PASCAL:

```
PROGRAM CONSLUZ (INPUT, OUTPUT);
{Calcula el consumo de energía y el importe a abonar}
VAR
        PKW, LACT, LANT, CONS, IMPOR : REAL;
BEGIN
  WRITE ('INGRESE PRECIO Y LECTURAS ACTUAL Y ANTERIOR');
  READLN (PKW, LACT, LANT);
  CONS := LACT - LANT;
  IMPOR := CONS * PKW;
  WRITELN (PKW:6:2, LACT:12:0, LANT:12:0);
  WRITELN;
  WRITE ('EL IMPORTE ES =':23, IMPOR:7:2)
END.
```

UNIDAD Nº 3

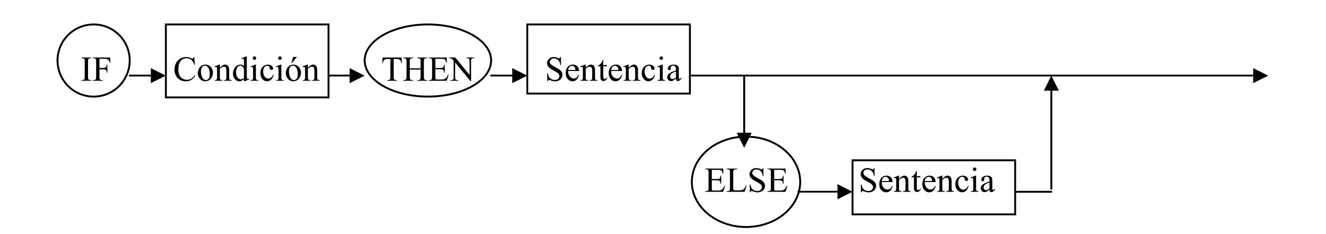
ESTRUCTURAS DE CONTROL

Hay tres estructuras básicas de control:

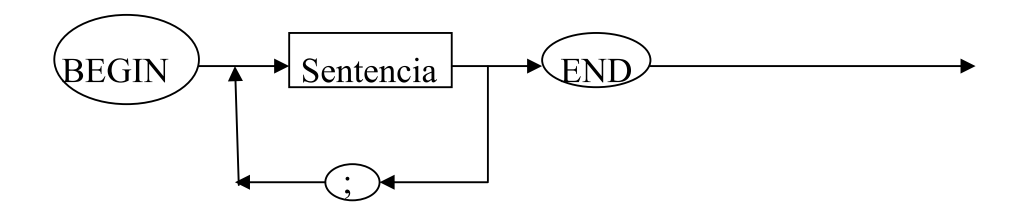
- LA SECUENCIAL (vista en las unidades anteriores)
- LA BIFURCACION O SELECCION
- LA REPETICION O ITERACION

3-1. LA BIFURCACION O SELECCION:

Hemos visto la selección entre dos caminos dependiendo de una condición. Esta sentencia es la sentencia IF , y su sintaxis es la siguiente:



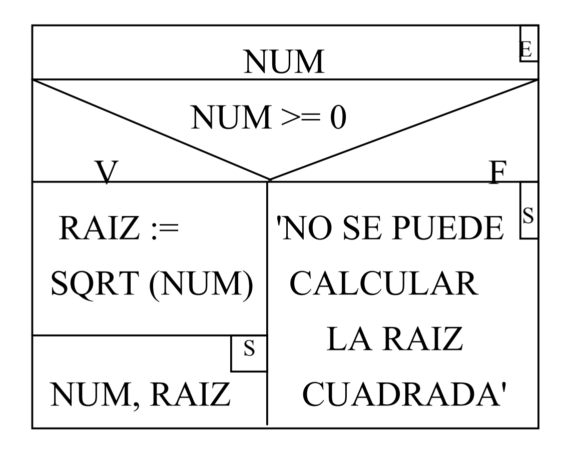
Donde dice Sentencia, puede tratarse de un grupo de sentencias, a este grupo se lo denomina: SENTENCIA COMPUESTA y va encerrado entre BEGIN y END ; la sintaxis de una sentencia compuesta es la siguiente:



3-1-1- EJEMPLOS:

1)- Dado un número real hallar, si es posible, su raíz cuadrada

a) DIAGRAMA DE CHAPIN:

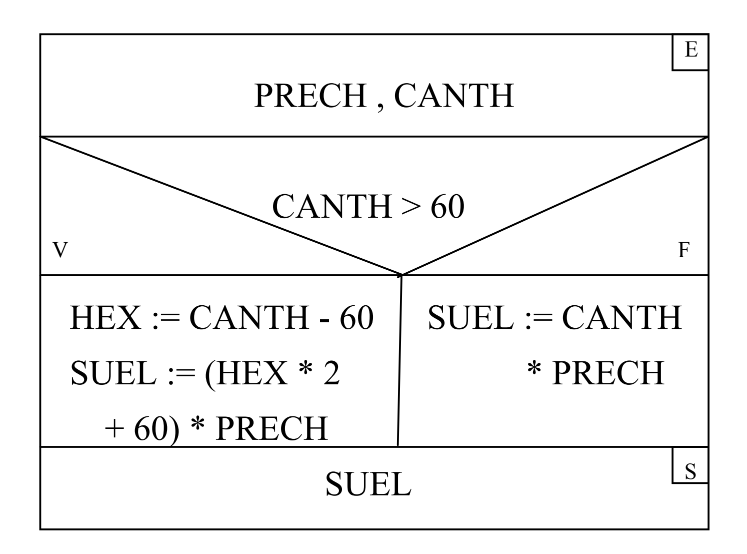


b) PROGRAMA EN PASCAL:

```
PROGRAM RAIZCUAD (Input, Output);
VAR
     NUM, RAIZ : REAL;
BEGIN
   WRITE ('INGRESE NUMERO');
   READLN (NUM);
   IF NUM >= 0
        THEN BEGIN
                 RAIZ := SQRT (NUM);
                 WRITE (NUM:10:2, RAIZ:10:4)
                 END
        ELSE WRITE ('NO SE PUEDE CALCULAR LA RAIZ CUADRADA')
END.
```

2)- A partir de los datos de: Pago por hora y Cantidad de horas trabajadas calcular el Sueldo de un operario, sabiendo que si las horas trabajadas superan 60, las excedentes se pagan el doble.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA EN PASCAL:

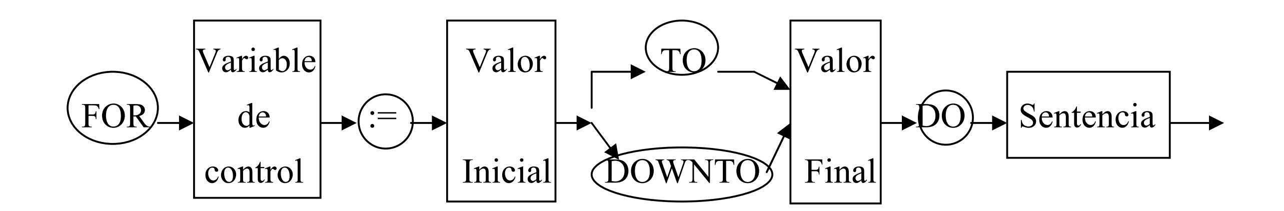
```
PROGRAM SUELDO (INPUT, OUTPUT);
VAR
     PRECH , SUEL : REAL;
     CANTH , HEX : INTEGER;
BEGIN
  WRITE ('INGRESE PRECIO-HORA Y CANT-HORAS TRABAJADAS);
  READLN ( PRECH, CANTH);
  IF CANTH > 60
     THEN BEGIN
               HEX := CANTH - 60;
               SUEL := (HEX * 2 + 60) * PRECH
               END
      ELSE SUEL := CANTH * PRECH;
```

```
WRITE (' EL SUELDO DEL OPERARIO ES DE $', SUEL:7:2)
END.
```

3-2. REPETICION O ITERACION:

3-2-1. CON CANTIDAD CONOCIDA DE VECES

Cuando una sentencia o un grupo de sentencias deben ejecutarse más de una vez utilizamos una estructura de repetición. Si sabemos qué cantidad de veces se van repetir, utilizamos la sentencia PARA; esta sentencia en PASCAL es la sentencia FOR, y su sintaxis es la siguiente:



La variable de control, el valor inicial y el valor final deben ser del mismo tipo. La variable de control debe declararse como cualquier otra variable.-

Donde dice Sentencia puede tratarse de una Sentencia Compuesta (cuya sintaxis ya la hemos visto).

TO y DOWNTO son palabras especiales. Se usa TO cuando la variable de control toma el valor inicial y llega al valor final incrementándose (sentencia FOR ascendente); de lo contrario se usa DOWNTO (sentencia FOR descendente).-

La ejecución de la sentencia FOR tiene lugar de la siguiente forma: primero se evalúan las expresiones (si así fuera) que dan el valor inicial y valor el final y se almacenan en memoria , y después se asigna a la variable de control el valor inicial.-

Se realiza entonces una comparación entre el valor de la variable de control y el final. Si el valor de la variable de control es mayor que el valor final (suponiendo que estamos usando TO y no DOWNTO), entonces ya no se ejecutan las sentencias del ciclo; en cuyo caso el control pasa a la sentencia siguiente de la última que constituye el rango del ciclo.-

Si por el contrario el valor de la variable de control es menor ó igual que el valor final entonces se ejecutan una vez más las sentencias que constituyen el ciclo, se incrementa la variable de control y se vuelve a comparar.-

El proceso continúa hasta que la variable de control toma un valor mayor que el valor final, en cuyo caso termina el ciclo.-

<u>Es importante destacar que:</u>

**** TODA SENTENCIA DENTRO DEL CICLO QUE INTENTE CAMBIAR EL VALOR DE LA VARIABLE DE CONTROL, DARA ERROR.-***

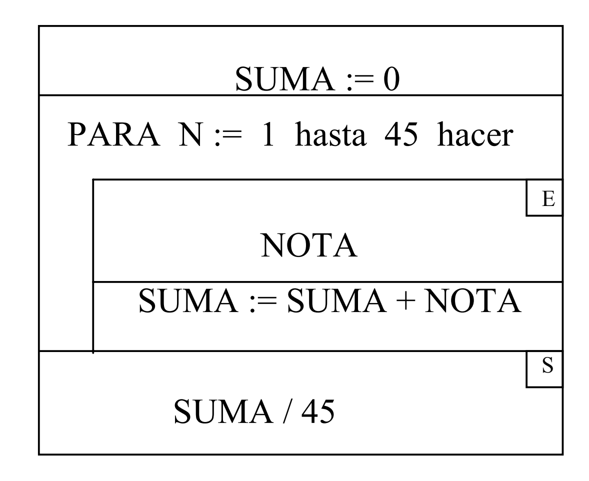
**** AL SALIR DEL CICLO LA VARIABLE DE CONTROL QUEDA CON VALOR INDEFINIDO.***

(No así dentro del ciclo, donde se puede utilizar su contenido).-

<u>EJEMPLOS:</u>

1)-Dadas las notas de un parcial de los 45 alumnos de un curso, se desea obtener la nota promedio del curso.-

a) <u>DIAGRAMA DE CHAPIN:</u>

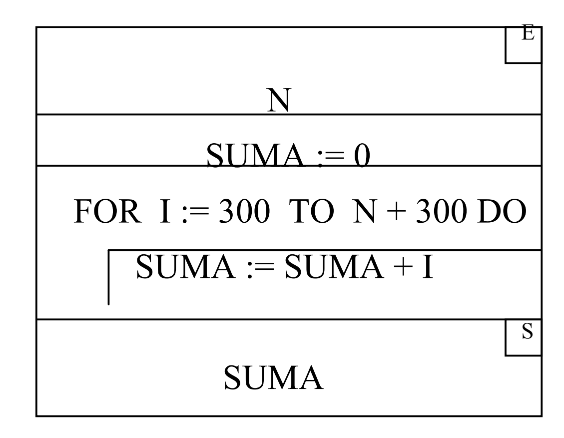


b) PROGRAMA EN PASCAL:

```
PROGRAM PROMEDIO (INPUT, OUTPUT);
VAR
      N : INTEGER;
      SUMA , NOTA : REAL;
BEGIN
  SUMA := 0;
  FOR N := 1 TO 45 DO
    BEGIN
      WRITE (' INGRESE NOTA ');
      READLN ( NOTA);
      SUMA := SUMA + NOTA
    END;
  WRITE (' EL PROMEDIO DEL CURSO ES = ', SUMA/45 :4:2)
END.
```

2)- Se desea obtener la suma de los N números naturales posteriores al número 300 inclusive.-

a) DIAGRAMA DE CHAPIN:



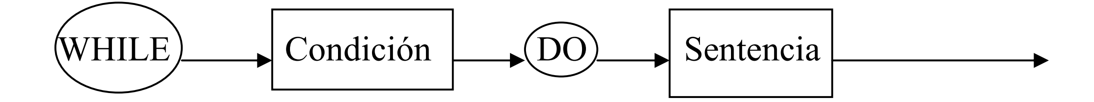
b) PROGRAMA EN PASCAL:

```
PROGRAM SUMANAT ( INPUT, OUTPUT);
VAR
    N , I : INTEGER;
    SUMA : REAL;
BEGIN
  WRITE ('INGRESE CANTIDAD DE NROS.');
  READLN (N);
  SUMA := 0;
  FOR I := 300 TO N + 300 DO
      SUMA := SUMA + I;
  WRITE ('LA SUMA DE LOS',N:4,'NROS NATURALES >= 300 ES',
        SUMA:10:0)
END.
```

3-2-2. CON CANTIDAD DESCONOCIDA DE VECES

Cuando una sentencia o un grupo de sentencias deben repetirse más de una vez, dependiendo de una condición, utilizamos la estructura MIENTRAS o REPETIR.-

3-2-2-1. La sentencia Mientras es la sentencia WHILE en PASCAL, y su sintaxis es la siguiente:



Donde dice Sentencia puede tratarse de una SENTENCIA COMPUESTA, como la que ya vimos.-

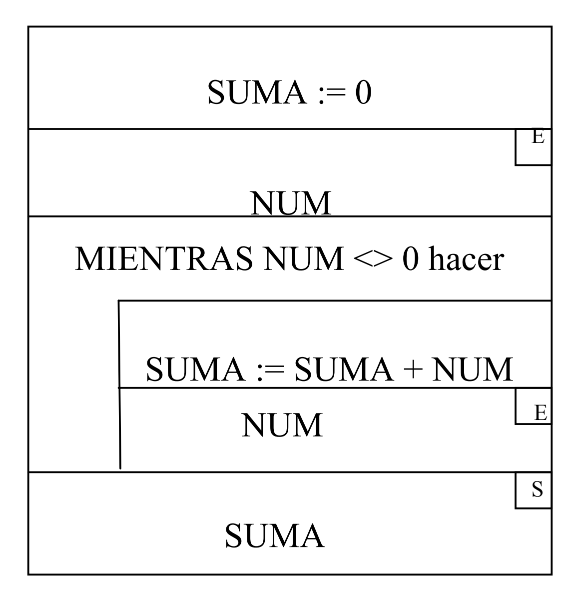
Mientras la Condición sea verdadera se ejecutan la ó las sentencias del ciclo. Cuando la condición sea falsa, el control pasa a la sentencia siguiente de la última que compone el ciclo.-

Por lo tanto si la condición es falsa la primera vez, pueden no ejecutarse nunca las sentencias del ciclo de repetición.-

EJEMPLOS:

1)-Se van ingresando números distintos de cero, salvo el último valor. Determinar su suma.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA EN PASCAL:

```
PROGRAM SUMANDO (INPUT, OUTPUT);
VAR
     NUM , SUMA : REAL;
 BEGIN
   SUMA := 0;
   WRITE ('INGRESE NUMERO');
   READLN (NUM);
   WHILE NUM <> 0 DO
     BEGIN
         SUMA := SUMA + NUM;
```

```
        WRITE ('INGRESE NUMERO');
        READLN (NUM)
      END;
    WRITE ('LA SUMA ES = ', SUMA:9:2)
  END.
```

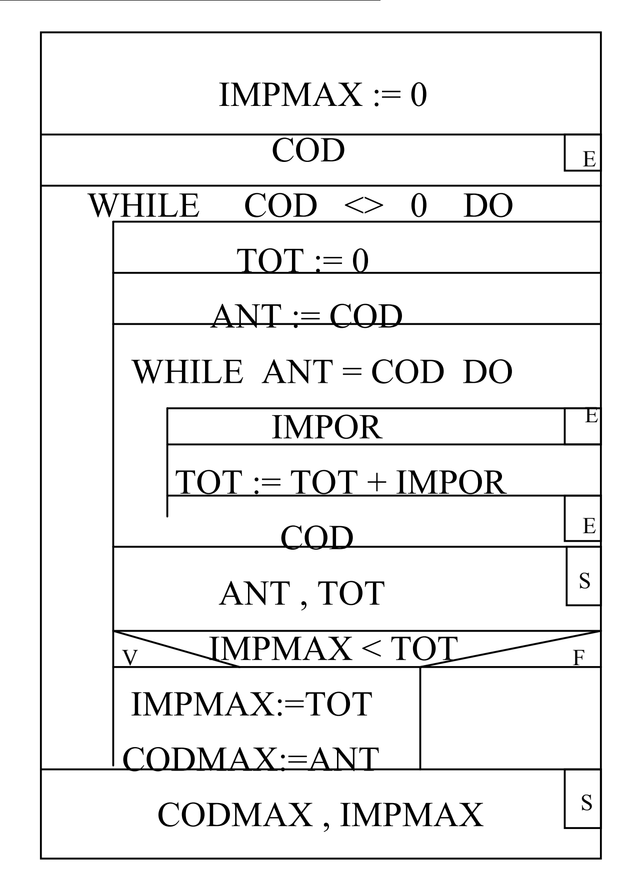
2)- Se desea saber el total de ventas de cada uno de los vendedores de una empresa. A tal fin se tienen como datos: el código de vendedor y el importe de cada una de las ventas; un vendedor puede haber realizado más de una venta. No se sabe la cantidad de vendedores que tiene la empresa ni la cantidad de ventas hechas por cada vendedor (un código de vendedor igual a cero es fin de datos).-

ESTOS DATOS ESTAN ORDENADOS POR CODIGO DE VENDEDOR

Exhibir cada código de vendedor y su total correspondiente y al final, el código de vendedor con mayor importe vendido y dicho importe.-

Resolverlo usando CORTE DE CONTROL.-

a) <u>DIAGRAMA DE CHAPIN:</u>



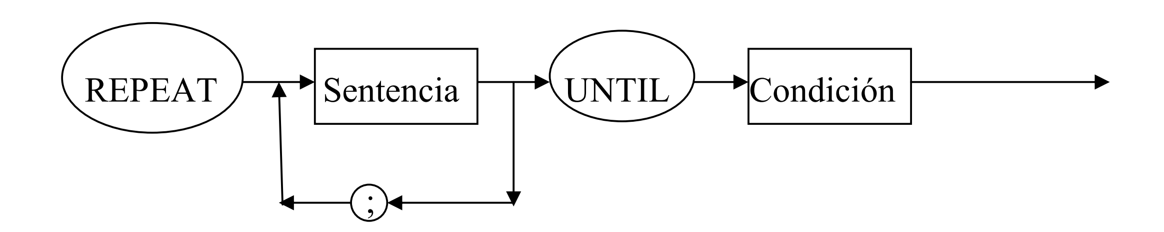
b) PROGRAMA EN PASCAL:

```
PROGRAM VENDEDOR (INPUT, OUTPUT);
    {EJEMPLO DE CORTE DE CONTROL}
VAR
     COD , ANT , CODMAX : INTEGER;
     IMPOR , TOT , IMPMAX : REAL;
BEGIN
  IMPMAX := 0;
  WRITE ('INGRESE CODIGO');
  READLN (COD);
 WHILE COD <> 0 DO
   BEGIN
      TOT := 0;
      ANT := COD;
      WHILE ANT = COD DO
       BEGIN
       WRITE ('INGRESE IMPORTE');
       READLN (IMPOR);
       TOT := TOT + IMPOR;
       WRITE ('INGRESE CODIGO');
       READLN (COD)
     END;
    WRITE ('EL VENDEDOR',ANT:4,' VENDIO $ ',TOT:15:2);
    IF TOT > IMPMAX
      THEN
        BEGIN
           IMPMAX := TOT;
           CODMAX := ANT
        END;
   END;
  WRITE ('EL VENDEDOR :',CODMAX:4,'TUVO MAYOR IMPORTE: $',
```

IMPMAX:15:2)

END.

3-2-2-2. La sentencia REPETIR es la sentencia REPEAT en PASCAL, y su sintaxis es la siguiente:

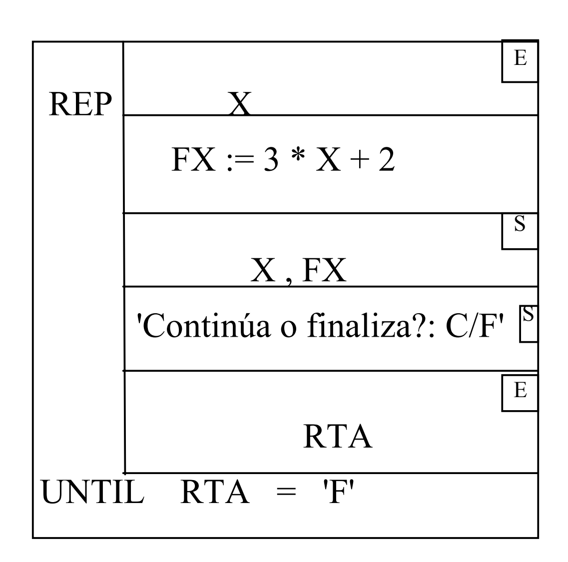


Repite la ejecución de la ó las sentencias del ciclo hasta que la condición sea verdadera. Por lo tanto el ciclo se ejecuta al menos una vez, pues compara al final del mismo. Esta es la gran diferencia que tiene con la sentencia WHILE.-

EJEMPLOS:

1)- Evaluar y tabular la función f(X) = 3X + 2 para diferentes valores de X .-

a) DIAGRAMA DE CHAPIN:

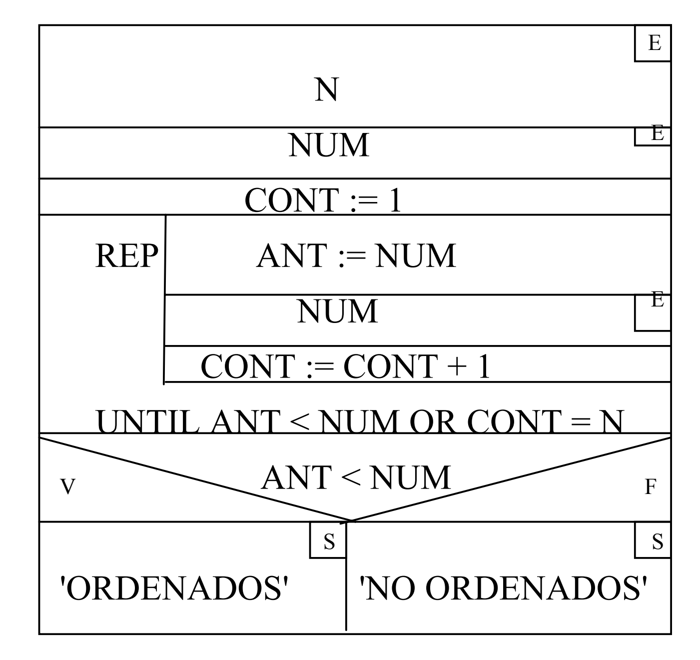


b) PROGRAMA EN PASCAL:

```
PROGRAM FUNCION (INPUT, OUTPUT);
 VAR
       X , FX : INTEGER;
       RTA : CHAR;
 BEGIN
   REPEAT
       WRITE ('INGRESE VALOR ');
       READLN (X);
       FX := 3 * X + 2;
       WRITE ('CONTINUA O FINALIZA INGRESANDO? C/F ');
       READLN (RTA)
   UNTIL RTA = 'F'
 END.
```

2)- Determinar si una lista de N números está ordenada de mayor a menor.-

a) DIAGRAMA DE CHAPIN:



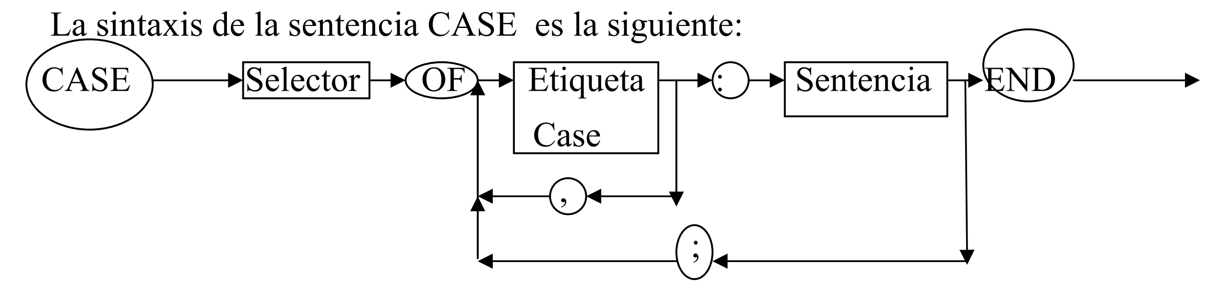
b) PROGRAMA EN PASCAL:

```
PROGRAM ORDEN (INPUT, OUTPUT);
VAR
    N , CONT , ANT , NUM : INTEGER;
BEGIN
  WRITE ('INGRESE LA CANTIDAD DE NROS.');
  READLN (N);
  WRITE ('INGRESE UN NRO.');
  READLN (NUM);
  CONT := 1;
  REPEAT
     ANT := NUM;
     WRITE ('INGRESE NRO.');
     READLN (NUM);
     CONT := CONT + 1
  UNTIL ANT < NUM OR CONT = N;
  IF ANT < NUM
     THEN WRITE ('NO ESTAN ORDENADOS')
     ELSE WRITE ('ESTAN ORDENADOS')
END.
```

3-3. SELECCION MULTIPLE:

Hemos visto también que si para optar por un camino a seguir no se depende de una condición, sino del valor que contenga un dato o expresión (el selector) podíamos utilizar la sentencia CASOS; esta sentencia es la sentencia CASE en PASCAL.-

La sintaxis de la sentencia CASE es la siguiente:



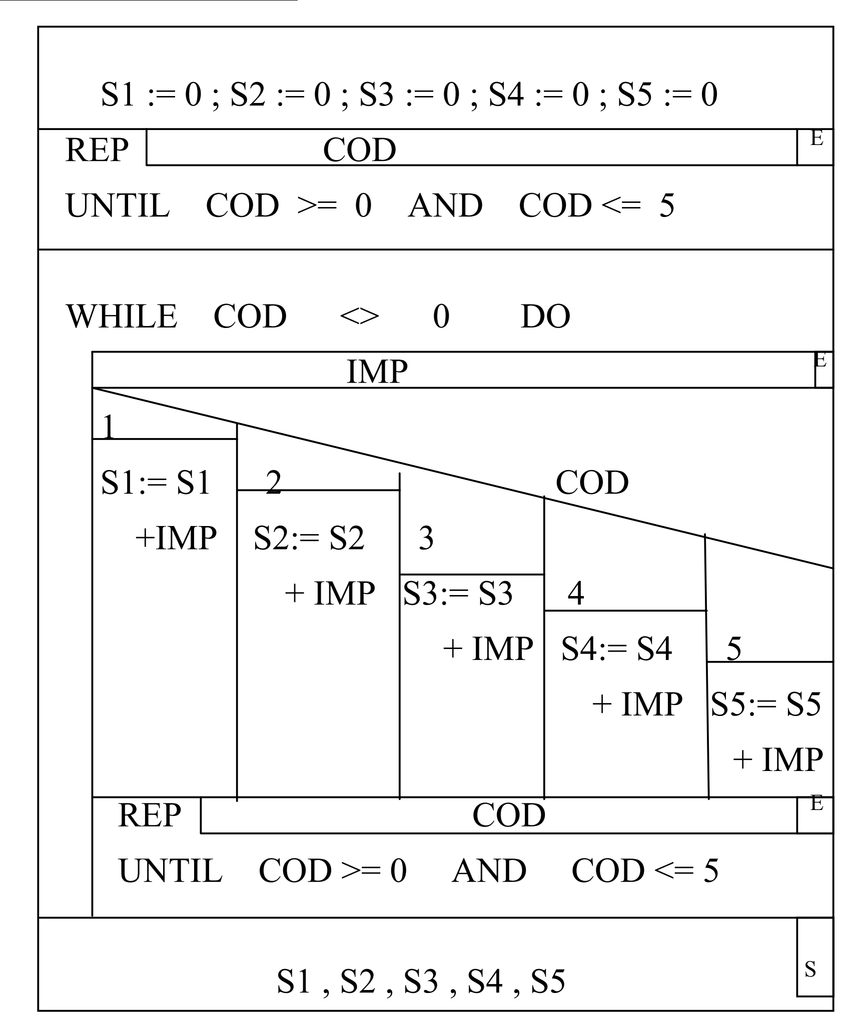
La sentencia CASE selecciona para su ejecución aquella sentencia cuya etiqueta sea igual al valor actual del selector. Si no existe la etiqueta el efecto es indefinido (NO TIENE SALIDA POR ERROR).-

Donde dice Sentencia puede tratarse de una SENTENCIA COMPUESTA.

EJEMPLO:

Se tienen como datos los importes de las ventas de cada una de las sucursales de una empresa, junto con el código de sucursal (1, 2, 3, 4 ó 5).- Cada sucursal puede tener varias ventas. Los datos no están ordenados por código de sucursal. Un código igual a cero indica fin de datos.- Obtener el total de ventas para cada sucursal.-

a) DIAGRAMA DE CHAPIN:



b) <u>PROGRAMA EN PASCAL:</u>

```
PROGRAM VENTAS (INPUT, OUTPUT);
VAR
    COD : INTEGER;
    S1 , S2 , S3 , S4 , S5 , IMP : REAL;
BEGIN
  S1:= 0; S2:= 0; S3:= 0; S4:= 0; S5:= 0;
  REPEAT
    WRITE ('INGRESE CODIGO');   READLN (COD)
  UNTIL ( COD >= 0 )  AND  ( COD <= 5 );
  WHILE COD <> 0 DO
    BEGIN
      WRITE ('INGRESE IMPORTE');
      READLN (IMP);
      CASE COD OF
          1 : S1 := S1 + IMP;
          2 : S2 := S2 + IMP;
          3 : S3 := S3 + IMP;
          4 : S4 := S4 + IMP;
          5 : S5 := S5 + IMP
      END;
      REPEAT
          WRITE ('INGRESE CODIGO');  READLN (COD)
      UNTIL ( COD >= 0 ) AND ( COD <= 5)
    END;
  WRITELN ('TOTAL SUCURSAL 1 :':30, S1:12:2);
  WRITELN ('TOTAL SUCURSAL 2 :':30, S2:12:2);
  WRITELN ('TOTAL SUCURSAL 3 :':30, S3:12:2);
  WRITELN ('TOTAL SUCURSAL 4 :':30, S4:12:2);
  WRITE ('TOTAL SUCURSAL 5 :':30, S5:12:2)
END.
```

UNIDAD Nº 4

TIPOS DE DATOS

4-1. TIPOS ENUMERADOS O ESCALARES:

PASCAL nos proporciona la manera de especificar nuestros propios tipos de datos.-

Los enumerados o escalares son aquellos en los que mencionamos cada uno de los valores que puede contener, mediante la definición:

```
TYPE T = ( C1, C2, ......, Cn );
```

EJEMPLO:

Si declaramos:

```
TYPE
    COLOR = (BLANCO, VERDE, ROJO);
    DIAS = (LUNES, MARTES MIERCOLES, JUEVES, VIERNES);
VAR
    D: DIAS;
    BAN: COLOR;
```

podemos escribir las siguientes sentencias:

```
D := MARTES;
BAN := BLANCO;
```

Si por error escribiéramos esta sentencia de asignación:

```
D:= SABADO;
```

nos marcaría el error.-

PASCAL también nos permite declarar específicamente las variables de esta manera:

```
VAR
    BAN1, BAN2: (BLANCO, VERDE, ROJO);
```

4-2. TIPOS SUBRANGO O INTERVALO:

Son los tipos de datos cuyos valores estén dentro de ciertos límites: LIMITE INFERIOR y LIMITE SUPERIOR.-

La forma general de definición es:

```
TYPE T = mín .. máx ;
```

siendo mín y máx, constantes.-

EJEMPLO:

Si declaramos:

```
TYPE
    NUM = 20 .. 100;
VAR
    EDAD: NUM;
```

podemos escribir la sentencia:

```
EDAD := 50;
```

Si pusiéramos por descuido o error:

```
EDAD := 150;
```

nos daría error.-

PASCAL también nos permite declarar las variables de esta manera:

```
VAR
      EDAD: 20 .. 100;
```

4-3. VARIABLES ESTRUCTURADAS: (ARREGLOS)

Hasta ahora hemos visto tipos de datos simples. Algunas veces es necesario almacenar y referenciar variables como un grupo. Estas estructuras de datos nos permiten escribir programas para manipular datos más fácilmente.-

Un ARRAY es un grupo de elementos a los que se les da un nombre común.-

Se accede a cada elemento por su posición dentro del grupo.-

Todos los elementos de un ARRAY deben ser del mismo tipo.-

4-3-1. ARRAYS UNIDIMENSIONALES (Vectores):

Un ARRAY unidimensional es un tipo de datos estructurados compuesto de un número fijo de componentes del mismo tipo, con cada componente directamente accesible mediante el índice.-

La forma general de declaración es:

```
VAR
      nombre : ARRAY [índice inferior .. índice superior] OF tipo;
```

Donde nombre es el nombre de la variable (cualquier identificador válido en PASCAL); índice inferior e índice superior indican el rango del ARRAY; y tipo es el tipo de datos de todos los elementos de la variable.-

Un elemento particular del ARRAY se representa por el nombre de la variable seguido de un índice encerrado entre corchetes, que indica la posición que ocupa ese elemento:

nombre [índice]

EJEMPLOS:

a) VAR

```
    NOM: ARRAY [1 .. 20] OF CHAR;
    CONT: ARRAY ['A' .. 'P'] OF INTEGER;
```

b) TYPE

```
    COD = 20..60;
VAR
    IMP: ARRAY [COD] OF REAL;
```

c) TYPE

```
    COLOR = (ROJO, VERDE, BLANCO, NEGRO, AZUL);
VAR
   BANDERA; ARRAY [1..30] OF COLOR;
```

Eventualmente puede hacer falta durante la ejecución de un programa, realizar asignaciones de todos los elementos de un array a otro; es decir, asignar a cada elemento de un array los elementos correspondientes de otro array ; para poder hacer esto, debemos declarar los dos arrays de "igual" manera, y escribir la sentencia de asignación con los nombres de los arrays solamente sin índice:

Por ejemplo si declaramos:

```
VAR  SUM, NUM1, NUM2: ARRAY [1..20] OF INTEGER;
```

Podemos escribir la siguiente sentencia de asignación:

```
NUM1 := NUM2;
```

PASCAL NO PERMITE:

a) REALIZAR OPERACIONES CON ARRAYS COMPLETOS TAL COMO:

```
SUM := NUM1 + NUM2;
```

b) ASIGNARLE UNA CONSTANTE A TODO UN ARRAY:

NUM2 := 0;

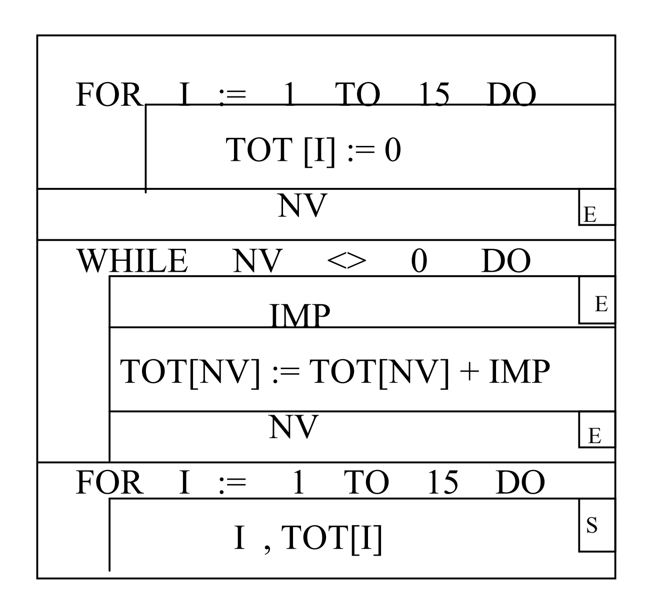
EJEMPLO:

Se tienen como datos: NRO. DE VENDEDOR e IMPORTE de cada una de las ventas realizadas por cada uno de los vendedores de una empresa.-

No se sabe cuántas ventas se han realizado por lo que un NRO. DE VENDEDOR igual a cero indica fin de datos.- Los vendedores están numerados del 1 al 15.

Se desea obtener el total vendido por cada vendedor.

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM VENTAS (INPUT, OUTPUT);
 VAR
      TOT : ARRAY [1..15] OF REAL;
      NV, I : INTEGER;
      IMP : REAL;
 BEGIN
```

```
  FOR I := 1 TO 15 DO
      TOT[I] := 0;
  WRITE ('INGRESE NRO VENDEDOR');
  READLN (NV);
  WHILE NV <> 0 DO
    BEGIN
      WRITE ('INGRESE IMPORTE');
      READLN (IMP);
      TOT[NV] := TOT[NV] + IMP;
      WRITE ('INGRESE NRO. VENDEDOR');
      READLN (NV)
    END;
  FOR I := 1 TO 15 DO
      WRITELN ('EL VENDEDOR NRO.':25, I:3, 'VENDIO $ ', TOT[I]:5:2)
END.
```

4-3-1-1. ORDENAMIENTO DE UN ARREGLO UNIDIMENSIONAL:

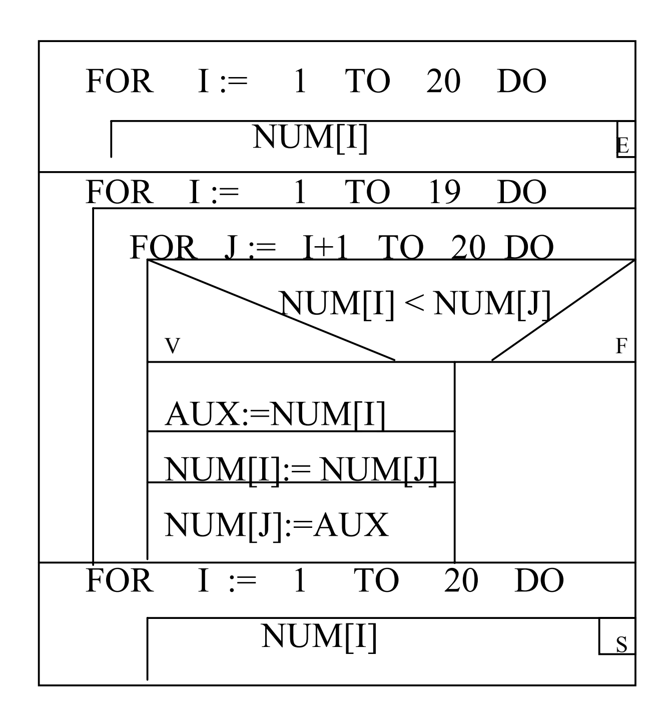
Para ordenar los elementos de un arreglo, ya sea en forma creciente o decreciente, existen varios métodos; nosotros veremos el método conocido por Falso Burbuja, que sin ser el más rápido en algunos casos, es uno de los más fáciles de entender.-

Este método consiste en comparar cada uno de los elementos del arreglo con todos los restantes y cambiarlos o no entre sí de acuerdo a la forma en que estemos ordenando el arreglo.-

EJEMPLO:

Ingresar 20 números enteros en un arreglo y luego mostrarlos ordenados en forma decreciente.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM ORDEN (INPUT, OUTPUT);
 VAR
    I, J, AUX : INTEGER;
    NUM : ARRAY [1..20] OF INTEGER;
 BEGIN
   FOR I := 1 TO 20 DO
      BEGIN
        WRITE ('INGRESE NRO.');
        READLN (NUM[I])
      END;
   FOR I := 1 TO 19 DO
       FOR J := I+1 TO 20 DO
          IF NUM[I] < NUM[J]
            THEN
                BEGIN
```

```
                    AUX := NUM[I];
                    NUM[I] := NUM[J];
                    NUM[J] := AUX
                END;
            FOR I := 1 TO 20 DO
                WRITE (NUM[I],' ')
        END.
```

4-3-1-2. BUSQUEDA DE UN VALOR EN UN ARREGLO ORDENADO:

Si tenemos un arreglo ordenado, ya sea en forma creciente o decreciente y tenemos que buscar un valor dentro del mismo, la forma más rápida para hacerlo es por medio de la Búsqueda Dicotómica.-

Para explicar este método, supongamos que los elementos del arreglo están ordenados en forma creciente procediéndose entonces, de la siguiente manera:

Primero se fijan los extremos del intervalo de búsqueda, que serán las posiciones que ocupan el primero y el último elemento del arreglo respectivamente.-

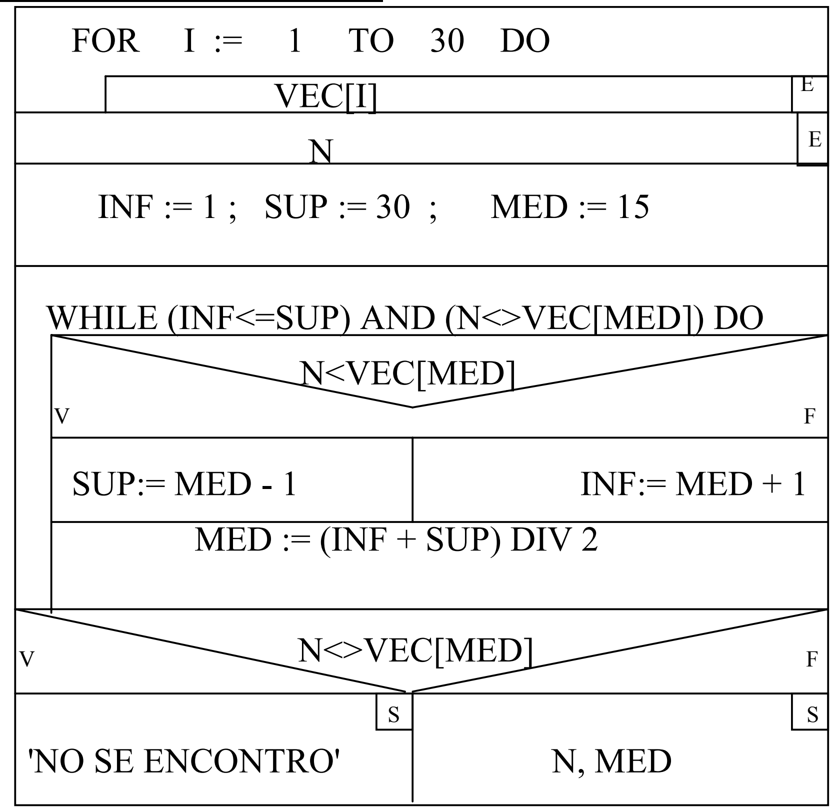
Luego se obtiene la posición media de dicho intervalo y se compara el valor a buscar con el contenido del arreglo en esa posición media; si el valor es mayor que dicho elemento se continúa la búsqueda en la segunda mitad del arreglo; si por el contrario el valor es menor, se continúa en la primera mitad y se halla nuevamente la posición media del nuevo intervalo de búsqueda.-

El proceso se repite hasta que se encuentra el valor a buscar, o bien hasta que no haya más intervalo de búsqueda, lo que significará que el valor buscado no se encuentra en el arreglo.-

EJEMPLO:

Se ingresan 30 números enteros ordenados en forma creciente y un valor N. Se desea saber si el valor N coincide con algún elemento del arreglo; si es así, indicar la posición en que fue encontrado, sino exhibir cartel aclaratorio.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM BUSQUEDA (INPUT, OUTPUT);
VAR
   I, INF, SUP, MED, N : INTEGER;
   VEC: ARRAY [1..30] OF INTEGER;
BEGIN
   FOR I := 1 TO 30 DO
     BEGIN
       WRITE ('INGRESE NRO.');
       READLN (VEC[I])
     END;
   WRITE ('INGRESE NRO. A BUSCAR');
   READLN (N);
   INF := 1;
   SUP := 30;
```

```
      MED := 15;
      WHILE (INF<=SUP) AND (N<>VEC[MED]) DO
        BEGIN
          IF N<VEC[MED]   THEN SUP:= MED - 1
                          ELSE INF:= MED + 1;
          MED := (INF + SUP) DIV 2
        END;
      IF N <> VEC[MED]
        THEN WRITE ('EL VALOR', N, 'NO FUE ENCONTRADO')
        ELSE WRITE ('EL VALOR',N,'SE ENCUENTRA EN LA POSICION', MED)
   END.
```

4-3-1-3. INTERCALACION DE ARRAYS ORDENADOS:

La intercalación de arrays es un proceso que consiste en tomar dos arrays ordenados, ya sea ambos en forma creciente o en forma decreciente, y obtener un array ordenado de igual manera que los dados.-

Para explicar este método, supongamos tener dos arrays ordenados en forma creciente cuyas dimensiones son n y m respectivamente, se obtendrá un nuevo array de dimensión (n + m), también ordenado en forma creciente, procediéndose de la siguiente manera:

Se comparan los primeros elementos de cada array, se selecciona el menor y se coloca en la primera posición del nuevo array.-

Luego se compara el elemento que quedó sin colocar, con el elemento que sigue del array cuyo elemento fue utilizado poniéndose el menor en la siguiente posición del nuevo array.-

Si dos elementos comparados son iguales, se coloca uno y luego el otro en el nuevo array y se pasan a comparar los elementos siguientes de cada uno de los dos arrays dados.-

El proceso continúa hasta que todos los elementos del primero o segundo array hayan sido utilizados, en cuyo caso los elementos restantes se agregan como están en el nuevo array.-

EJEMPLO:

Ingresar 10 números enteros ordenados en forma creciente en un array; luego 13 números enteros, también ordenados en forma creciente en otro array.-

Obtener por intercalación, un tercer array ordenado en forma creciente, y luego mostrarlo.-

a) DIAGRAMA DE CHAPIN:

```
FOR I := 1 TO 10 DO
    V1 [I]                                   E
FOR J := 1 TO 13 DO
    V2 [J]                                   E
I := 1 ; J := 1 ; K := 1
WHILE (I <= 10) AND (J <= 13) DO
    V1[I] < V2[J]
    V                                        F
    V3[K] := V1[I]      V1[I] = V2[J]
    I:= I + 1           V                    F
                        V3[K] :=V1[I]    V3[K] := V2[J]
                        I := I+1         J:= J + 1
                        K := K+1
                        V3[K] := V2[J]
                        J := J+1
    K := K + 1
I > 10
V                                            F
FOR L := J TO 13 DO          FOR L := I TO 10 DO
    V3[K] := V2[L]               V3[K] := V1[L]
    K := K + 1                   K := K + 1
FOR I := 1 TO 23 DO
    V3[I]                                    S
```

b) PROGRAMA PASCAL:

```
PROGRAM MERGE (INPUT, OUTPUT);
 VAR
      I, J, K, L : INTEGER;
      V1 : ARRAY [1..10] OF INTEGER;
      V2 : ARRAY [1..13] OF INTEGER;
      V3 : ARRAY [1..23] OF INTEGER;
 BEGIN
   FOR I := 1 TO 10 DO
     BEGIN
         WRITE ('INGRESE NRO.');
          READLN (V1[I])
     END;
   FOR J := 1 TO 13 DO
     BEGIN
         WRITE ('INGRESE NRO.');
          READLN (V2[J])
     END;
   I := 1;  J := 1;  K := 1;
   WHILE ( I<= 10 ) AND ( J <= 13 ) DO
     BEGIN
        IF V1[I] < V2[J]
            THEN BEGIN
                         V3[K] := V1[I];
                          I := I + 1
                      END
            ELSE IF V1[I] = V2[J]
                        THEN BEGIN
                                    V3[K] := V1[I];
                                     I := I + 1;
                                     K := K + 1;
```

```
                    V3[K] := V2[J];
                     J := J + 1
                   END
             ELSE BEGIN
                    V3[K] := V2[J];
                     J := J + 1
                  END;
   K := K + 1
  END;
 IF I > 10
  THEN  FOR L := J TO 13 DO
          BEGIN
            V3[K] := V2[L];
            K := K + 1
          END
  ELSE  FOR L := I TO 10 DO
          BEGIN
            V3[K] := V1[L];
            K := K + 1
          END;
 WRITELN (' ARREGLO ORDENADO ');
 FOR I := 1 TO 23 DO
    WRITE (V3[I], ' ')
END.
```

4-3-2. ARRAYS BIDIMENSIONALES (matrices):

Un array bidimensional es también un tipo de datos estructurados compuesto de un número fijo de componentes del mismo tipo, accediéndose a cada una de ellas mediante dos subíndices, el primero indica el número de fila y el segundo el número de columna en donde está dicho elemento.-

La forma general de declaración es :

VAR

nombre: ARRAY [índ inf .. índ sup , índ inf .. índ sup] OF tipo;

Un elemento particular del ARRAY se representa por:

nombre [índice fila , índice columna]

EJEMPLO:

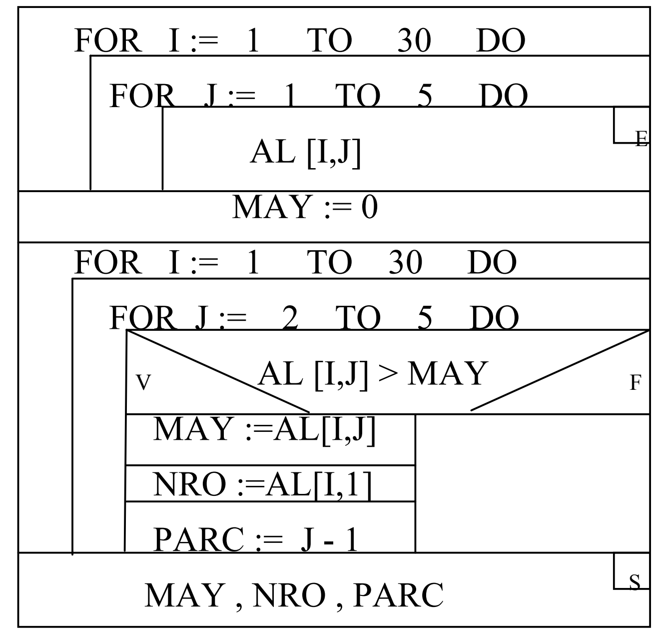
Se tienen los siguientes datos de 30 alumnos de primer año:

NRO. ALUMNO (entero)

NOTA 1, NOTA 2, NOTA 3, NOTA 4 (enteras)

Se desea cargar los datos en un arreglo bidimensional y luego exhibir el nro. de alumno que tuvo la mejor nota de todas y en qué parcial la obtuvo.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM PARCIAL (INPUT, OUTPUT);
 VAR
    I, J, MAY, NRO, PARC : INTEGER;
    AL : ARRAY [1..30, 1..5] OF INTEGER;
 BEGIN
  FOR I := 1 TO 30 DO
    BEGIN
      WRITE ('INGRESE NRO. DEL ALUMNO Y LAS 4 NOTAS');
      FOR J := 1 TO 5 DO
         READ (AL [I,J]);
      WRITELN
    END;
  MAY := 0;
  FOR I := 1 TO 30 DO
     FOR J := 2 TO 5 DO
        IF AL [I,J] > MAY
            THEN BEGIN
                   MAY := AL [I,J];
                   NRO := AL [I,1];
                   PARC := J - 1
                 END;
  WRITE ('LA MAYOR NOTA: ',MAY,' CORRESPONDE AL ALUMNO: ',
            NRO, 'EN EL PARCIAL NUMERO: ',PARC)
 END.
```

4-3-2-1. ORDENAMIENTO DE UN ARRAY BIDIMENSIONAL:

Tenemos que tener en cuenta que los datos que se cargan en un arreglo bidimensional responden, por fila o por columna, a una persona, a una agencia, a una comisión, etc.; por lo tanto, lo mismo que en una matriz matemática, no podemos

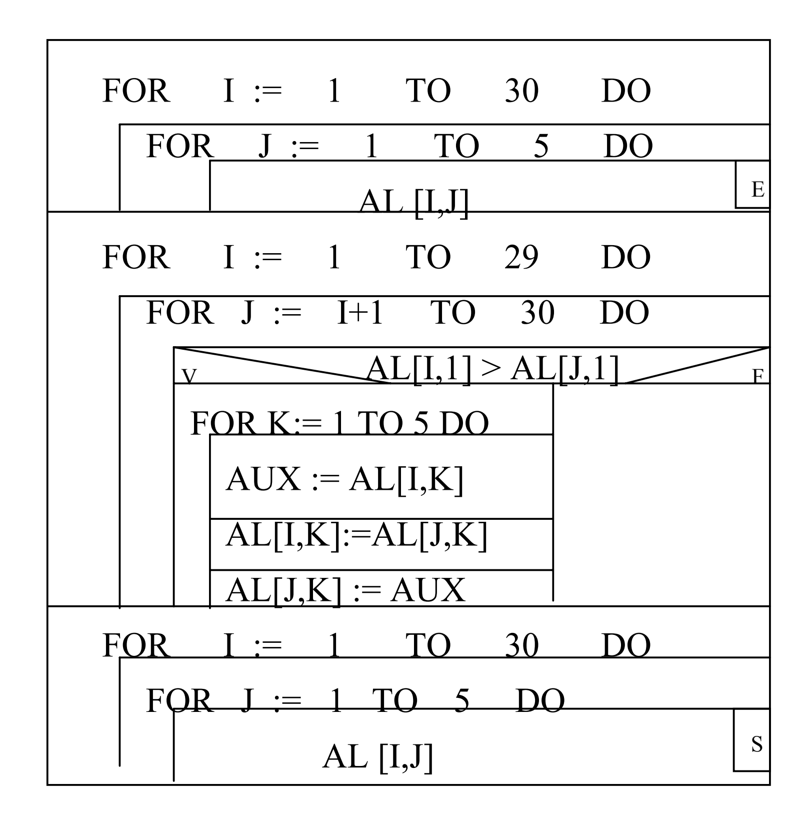
cambiar un elemento por otro de lugar; lo que sí podemos hacer es cambiar una fila o una columna por otra. Luego, un ordenamiento en un arreglo bidimensional tiene sentido, si se pide que se ordene el mismo de manera tal que los elementos de una fila o columna queden ordenados en forma creciente o decreciente.-

EJEMPLO:

Se tienen los mismos datos del ejercicio anterior.

Se desea un listado de los mismos ordenados en forma creciente por NRO. DE ALUMNO.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM LISTADO (INPUT, OUTPUT);
  VAR
       I, J, K, AUX : INTEGER;
```

```
  AL : ARRAY [1..30,1..5] OF INTEGER;

BEGIN
  WRITELN ('INGRESE EL NRO Y 4 NOTAS DE LOS 30 ALUMNOS');
  FOR I := 1 TO 30 DO
    BEGIN
      FOR J := 1 TO 5 DO
        READ ( AL[I,J]);
      WRITELN
    END;
  FOR I := 1 TO 29 DO
    FOR J := I+1 TO 30 DO
       IF AL[I,1] > AL[J,1]
         THEN
           FOR K := 1 TO 5 DO
             BEGIN
               AUX := AL[I,K];
               AL[I,K] := AL[J,K];
               AL[J,K] := AUX
             END;
  WRITELN ('    ALUMNO  NOTA 1  NOTA 2  NOTA 3  NOTA 4');
  FOR I := 1 TO 30 DO
    BEGIN
      FOR J := 1 TO 5 DO
        WRITE ( AL[I,J] : 8);
      WRITELN
    END
END.
```

4-3-2-2. BUSQUEDA EN UN ARREGLO BIDIMENSIONAL:

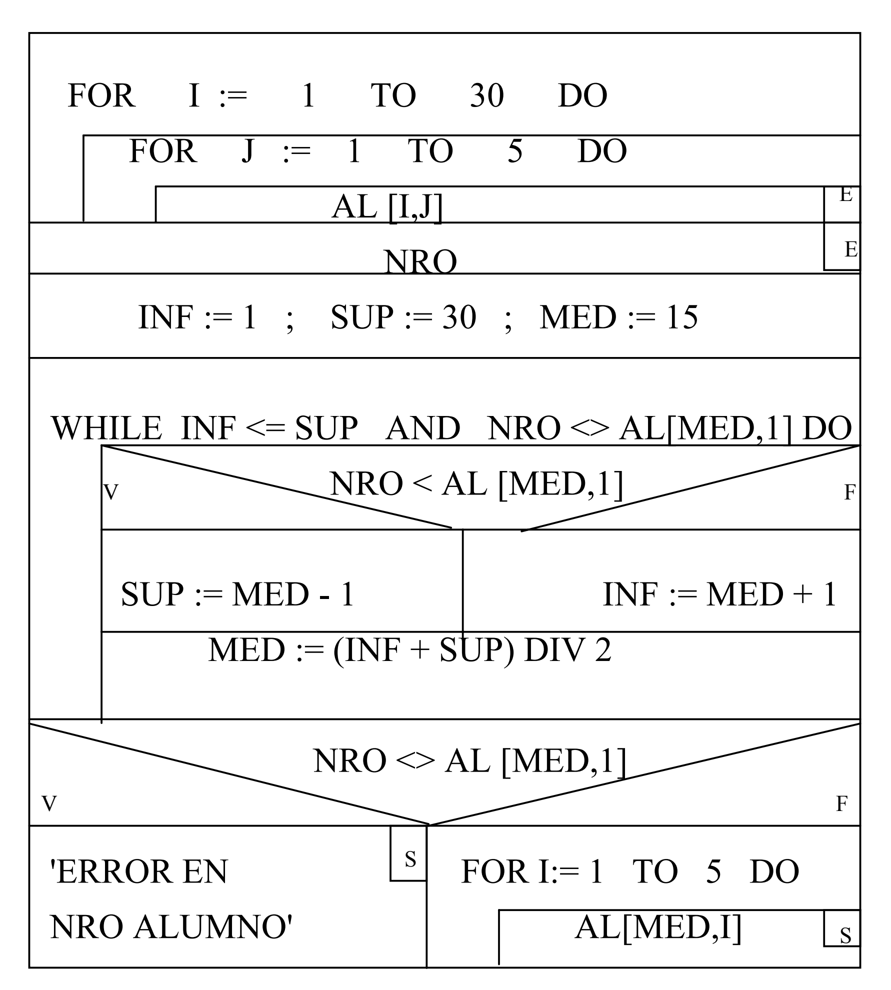
Si los elementos de un arreglo bidimensional están ordenados por una columna o una fila, ya sea en forma creciente o decreciente, se puede buscar un valor dentro de dicha fila o columna por medio de la búsqueda dicotómica.-

EJEMPLO:

Se tiene los datos de los 30 alumnos del ejercicio anterior ya ordenados en forma creciente por NRO DE ALUMNO.

Se desea ingresar un NRO DE ALUMNO y buscarlo por medio de la búsqueda dicotómica dentro del arreglo. Si se encuentra, dar el nro de alumno junto con las notas de los 4 parciales; sino exhibir cartel aclaratorio.-

a) DIAGRAMA DE CHAPIN:



b) PROGRAMA PASCAL:

```
PROGRAM BUSQUEDA (INPUT, OUTPUT);
VAR
    I, J, INF, SUP, MED, NRO : INTEGER;
    AL : ARRAY [1..30, 1..5] OF INTEGER;
BEGIN
  WRITE ('INGRESE NRO Y 4 NOTAS DE LOS 30 ALUMNOS');
  FOR I := 1 TO 30 DO
     BEGIN
        FOR J := 1 TO 5 DO
            READ (AL[I,J]); WRITELN
     END;
  WRITE ('INGRESE NRO. DEL ALUMNO A BUSCAR');
  READLN (NRO);
  INF := 1; SUP := 30; MED := 15;
  WHILE INF <= SUP AND NRO <> AL[MED,1] DO
    BEGIN
      IF NRO < AL[MED,1]
         THEN SUP := MED - 1
         ELSE INF := MED + 1;
      MED := (INF + SUP) DIV 2
    END;
  IF NRO <> AL [MED,1]
      THEN WRITE (' ERROR EN EL NRO. DE ALUMNO')
      ELSE
          BEGIN
             WRITELN ('  ALUMNO  NOTA 1  NOTA 2  NOTA 3  NOTA 4');
             FOR I := 1 TO 5 DO
                 WRITE (AL[MED,I]
          END
END.
```

UNIDAD Nº5

SUBPROGRAMAS

5 -1. INTRODUCCION

Es frecuente en programación que un grupo de sentencias deba repetirse varias veces con distintos datos, o sea que debamos escribirlas varias veces. PASCAL permite escribirlas una sola vez bajo la forma de subprogramas y usarlas las veces que sea necesario.-

Además, si un grupo de sentencias realiza una tarea específica, puede estar justificado el aislarlas formando un subprograma, aunque se las use una sola vez.-

Hay dos tipos de subprogramas: ***FUNCIONES y PROCEDIMIENTOS***.-

5-1-1. FUNCIONES:

Una función PASCAL es un grupo de sentencias dentro de un programa que forman un bloque (un subprograma) que realiza un número determinado de operaciones sobre un conjunto de argumentos dado y devuelve un **"solo valor".-**

Cada vez que se llama a la función, se transfiere el control al bloque de sentencias definidas por esa función.

Después que las sentencias han sido ejecutadas, el control vuelve a la sentencia en que fue llamada la función.-

La invocación de una función es de la forma:

nombre (argumento1, argumento2,....)

donde nombre, que es el nombre de la función, es un identificador válido en PASCAL y es donde vuelve el resultado; y cada argumento puede ser cualquier variable válida, constante o expresión.-

Esta invocación debe ser asignada a una variable, formar parte de una expresión asignada a una variable, puede estar en un write o en un if.-

Una definición de función tiene la forma:

FUNCTION ***nombre (declaración de parámetros): tipo;***

declaración de identificadores locales

Begin

sentencias ejecutables

End;

donde nombre es el nombre de la función; declaración de parámetros, contiene los parámetros (cada uno de los cuales debe ser un identificador válido en PASCAL) de la función y los tipos de datos que se asocian con cada uno de ellos; y tipo es el

tipo de resultado que devuelve la función.-

El orden de la lista de parámetros es el orden de correspondencia de dichos parámetros con la lista de argumentos de la llamada. Por lo tanto en número de parámetros y de argumentos debe ser el mismo, y el tipo de los que se correspondan debe coincidir.-

Si una función no necesita pasar parámetros, entonces se omiten las listas de parámetros y de argumentos.-

En la lista de parámetros sólo se permiten identificadores de tipo standard o de un tipo definido por el programador que debe declararse en el programa principal (o en el que hace la llamada).-

En la declaración de identificadores locales van las declaraciones de las variables que no son parámetros, usadas por la función.-

Las sentencias propiamente dichas de la función (cuerpo de la función) van entre Begin y End; , como ocurre con el programa principal.-

Las definiciones de función siempre tienen lugar después de la sección de declaración de variables, pero antes del cuerpo del programa en el cual es invocada dicha función.-

EJEMPLO:

Escribir un programa que calcule la expresión : $\sum_{i=0}^{n} x^{i}$

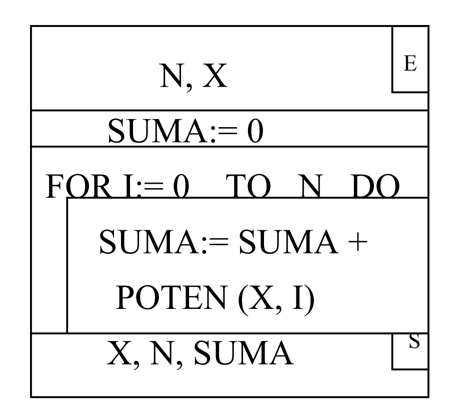
para cualquier par de valores n y x.-

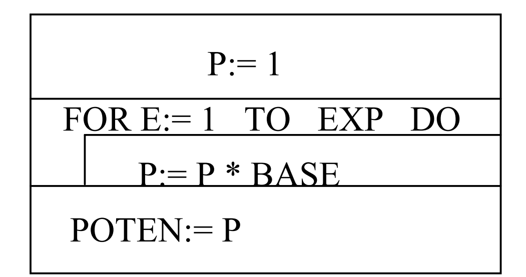
Para evaluar cada uno de los términos de la sumatoria, crear y utilizar una función llamada POTEN, que tenga como parámetros la base x y el exponente i.-

Exhibir: x, n y el resultado de la sumatoria.-

a) DIAGRAMA DE CHAPIN:

PROG. PPAL. FUNCTION POTEN (BASE: real; EXP: integer): real;





b) PROGRAMA PASCAL:

```
PROGRAM SUMATORI (INPUT, OUTPUT);
 VAR
     N, I: INTEGER;
     X, SUMA: REAL;
FUNCTION POTEN (BASE: REAL; EXP: INTEGER) : REAL;
```

```
 VAR
     P: REAL;
     E: INTEGER;
BEGIN
    P:= 1;
    FOR E:= 1 TO EXP DO
       P:= P * BASE;
    POTEN:= P
END;
BEGIN
   WRITE ('INGRESE EXTREMO SUMATORIA Y NRO.');
   READLN (N, X);
   SUMA:= 0;
   FOR I:= 0 TO N DO
     SUMA:= SUMA + POTEN(X, I);
   WRITELN ('LA SUMATORIA DE LOS TERMINOS DE BASE',X:3:2);
   WRITELN ('DESDE POTENCIA 0 A POTENCIA', N);
   WRITE ('ES IGUAL A', SUMA:10:2)
END.
```

5-1-2- PROCEDIMIENTOS:

Los procedimientos son subprogramas similares a las funciones pero con dos diferencias importantes.

La llamada a un procedimiento es similar a la de la función:

nombre (argumento1, argumento2,...)

pero esta debe estar sola, es decir no puede estar formando parte de expresiones, ni asignada a una variable, ni en un write, ni en un if. (primer diferencia)

La segunda diferencia es que con el procedimiento no se devuelve un valor al punto de llamada.

Un procedimiento está compuesto de un grupo de sentencias a las que se asigna un nombre (*identificador*) y constituye una unidad del programa. La tarea asignada al procedimiento se ejecuta siempre que Pascal encuentra el nombre del procedimiento.

La definición de un procedimiento en PASCAL es muy similar a la de una función, salvo algunas diferencias. Primero, la palabra clave es **PROCEDURE** en lugar de **FUNCTION** en la cabecera, y segundo que no contiene ningún atributo detrás de las declaraciones de los parámetros puesto que no vuelve ningún valor en el nombre del procedimiento.-

Una definición de procedimiento tiene la forma:

PROCEDURE ***nombre (declaración de parámetros);***

declaración de identificadores locales

Begin

sentencias ejecutables

End;

Si un parámetro tiene que modificar el valor de su argumento correspondiente, dicho parámetro debe ir precedido por la palabra **VAR**. En ausencia de VAR, cualquier cambio hecho al valor del parámetro no será reflejado en su argumento.-

Tanto en procedimientos como en funciones, como ya dijimos, las variables que son utilizadas por ellos van declaradas en la parte de declaración de ***variables locales*** y se las denomina de esa manera.-

Las variables declaradas en un programa que contenga funciones o procedimientos, son válidas (pueden utilizarse) en cualquier parte del programa, también dentro de las funciones y los procedimientos; a estas variables se las denomina ***variables globales***.

Si durante la ejecución de un subprograma se usa y se altera el valor de una variable global, al terminar el subprograma, el valor de la variable corresponde al último valor y no al original que tenía antes de que se ejecutara el subprograma.-

5 -2. CORRESPONDENCIA ARGUMENTO-PARAMETRO:

Como ya hemos visto, cada vez que se llama a una función o a un procedimiento, se establece una correspondencia entre cada argumento y su parámetro.-

Esta llamada puede ser:

***por valor* o *por referencia*.-**

5-2-1. LLAMADA POR VALOR:

Es la asignación del valor del argumento a su parámetro. El parámetro es entonces una variable independiente, de nueva creación que recibe el valor del argumento al comienzo de la ejecución del subprograma.

Puesto que parámetro y argumento son variables independientes, cualquier cambio que se produzca en el parámetro durante la ejecución del subprograma no tiene efecto en el valor del argumento.

Por lo tanto cuando se utiliza la llamada por valor, es imposible la devolución de valores al punto de llamada por medio de los parámetros; pues al terminar la ejecución del subprograma, la variable parámetro se destruye y el valor que contenía se pierde.-

5-2-2. LA LLAMADA POR REFERENCIA:

En el caso de que se requiera que el valor de una variable sea modificado por el subprograma invocado, debe hacerse el paso de parámetro por referencia, por medio del cual el subprograma invocado tiene acceso a la dirección en que se guarda el valor a modificar.

No implica la creación de una posición aparte de memoria para el parámetro, sino

mas bien produce el paso de la dirección de memoria donde se almacena el valor del argumento.-

El parámetro, en efecto, viene a ser simplemente otro nombre (o el mismo) para la misma dirección ya creada para el valor del argumento.

En este tipo de llamada, los parámetros van precedidos por la palabra **VAR**.-

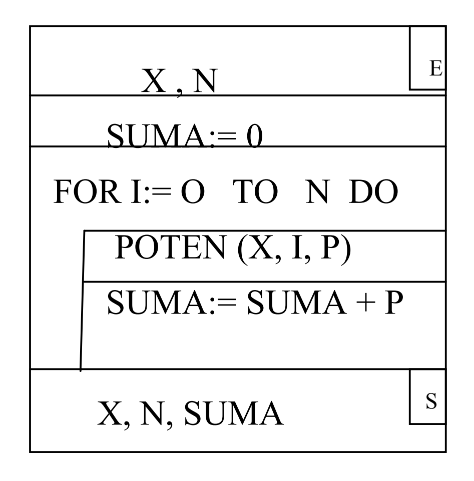
EJEMPLO:

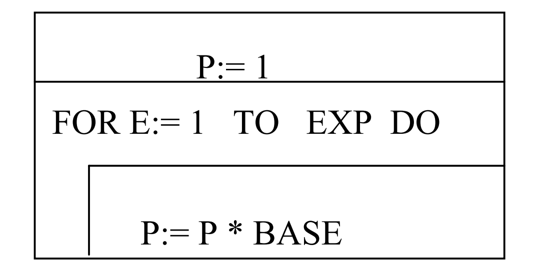
Realicemos el ejemplo anterior, pero en lugar de utilizar una función, utilicemos un Procedimiento llamado POTEN.-

a) DIAGRAMA DE CHAPIN:

PROG. PPAL.

PROCEDURE POTEN (BASE:real; EXP:integer; var P:real)





b) PROGRAMA PASCAL:

```
PROGRAM SUMATORI (INPUT, OUTPUT);
 VAR
     N, I: INTEGER;
     X, SUMA, P: REAL;
```

```
PROCEDURE POTEN (BASE: REAL; EXP: INTEGER; VAR P: REAL);
 VAR
     E: INTEGER;
 BEGIN
     P:= 1;
     FOR E:= 1 TO EXP DO
         P:= P * BASE
 END;
BEGIN
    WRITE ('INGRESE EXTREMO SUMATORIA Y NRO.');
    READLN (N, X);
    SUMA:= 0;
    FOR I:= 0 TO N DO
     BEGIN
       POTEN (X, I, P);
       SUMA:= SUMA + P
     END;
    WRITELN ('LA SUMATORIA DE LOS TERMINOS DE BASE',X:3:2);
    WRITELN ('DESDE POTENCIA 0 HASTA POTENCIA', N);
    WRITE ('ES IGUAL A', SUMA:10:2)
  END.
```

5-3- <u>RECURSIVIDAD</u>

En Pacal, a un procedimiento o función le es permitido no sólo invocar a otro procedimiento o función, sino también invocarse a sí mismo. Una invocación de éste tipo se dice que es *recursiva*.

La *función recursiva* más utilizada como ejemplo es la que calcula el factorial de un número entero no negativo, partiendo de las siguientes definiciones :

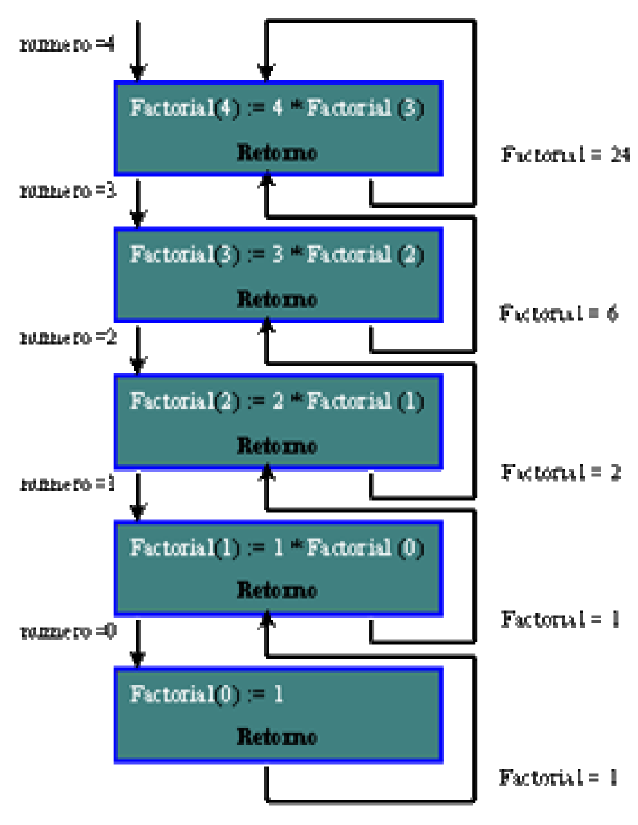
factorial (0) = 1

factorial (n) = n*factorial(n-1), para n>0

La función, escrita en Pascal, queda de la siguiente manera :

```
function factorial(numero:integer):integer;
begin
  if numero = 0 then
    factorial := 1
  else
    factorial := numero * factorial(numero-1)
end;
```

Si numero = 4, la función realiza los siguientes pasos :



1. factorial(4) := 4 * factorial(3) Se invoca a si misma y crea una segunda variable cuyo nombre es numero y su valor es igual a 3.
2. factorial(3) := 3 * factorial(2) Se invoca a si misma y crea una tercera variable cuyo nombre es numero y su valor es igual a 2.
3. factorial(2) := 2 * factorial(1) Se invoca a si misma y crea una cuarta variable cuyo nombre es numero y su valor es igual a 1.
4. factorial(1) := 1 * factorial(0) Se invoca a si misma y crea una quinta variable cuyo nombre es numero y su valor es igual a 0.
5. Como factorial(0) := 1, con éste valor se regresa a completar la invocación: factorial(1) := 1 * 1 , por lo que factorial(1) := 1

6. Con factorial(1) := 1, se regresa a completar: factorial(2) := 2 * 1, por lo que factorial(2) := 2
7. Con factorial(2) := 2, se regresa a completar : factorial(3) := 3 * 2, por lo que factorial(3) := 6
8. Con factorial(3) := 6, se regresa a completar : factorial(4) := 4 * 6, por lo que factorial(4) := 24 y éste será el valor que la función factorial devolverá al módulo que la haya invocado con un valor de parámetro local igual a 4 .

Otro ejemplo de procedimiento recursivo es el siguiente:

Supóngase que una persona se mete a una piscina cuya profundidad es de 5 metros. Su intención es tocar el fondo de la piscina y después salir a la superficie. Tanto en el descenso como en el ascenso se le va indicando la distancia desde la superficie (a cada metro).

```
Program Piscina;
Const
  prof_max = 5;
Var
  profundidad:integer;
procedure zambullida(Var profun :integer);
begin
  WriteLn('BAJA 1 PROFUNDIDAD = ',profun);
  profun := profun + 1;
  if profun <= prof_max then
    zambullida(profun)
  else
    WriteLn;
  profun := profun - 1;
  WriteLn('SUBE 1 PROFUNDIDAD = ', profun-1)
end;
begin
  profundidad := 1;
  zambullida(profundidad)
end.
```

UNIDAD Nº 6

OTRAS ESTRUCTURAS DE DATOS

6-1. REGISTROS:

Hasta ahora, la única estructura de datos que hemos visto son los arrays. Los registros son similares a los arrays pues también representan un grupo de elementos con un nombre común. Sin embargo, mientras que los elementos de un array deben ser todos del mismo tipo, los elementos de un registro pueden ser de distintos tipos de datos.

O sea, los registros son un tipo de datos estructurado (ó variable compuesta), con un número fijo de componentes (no necesariamente del mismo tipo) a las que se accede por el nombre, no por un subíndice.-

La declaración en Pascal de este tipo de datos es:

```
TYPE  nombre = RECORD
                    identificador del campo: tipo;
                       .
                       .
                    identificador del campo: tipo
                  END;
```

donde:

nombre es un identificador válido en Pascal

identificador del campo, es el nombre de una componente del registro

tipo, es el tipo de elemento de cada campo

Ejemplo:

```
TYPE ALUMNO = RECORD
                  NOMBRE: STRING  [20] ;
                  LEGAJO: REAL;
                  DNI: REAL;
                  NOTAS: ARRAY (1..6) OF INTEGER;
                  ACURSA: INTEGER
                END;
 VAR

     ESTUD: ALUMNO;
```

Para referirnos a un campo del registro especificamos el nombre de la variable y el nombre del campo separados por un punto.

Ejemplo: ESTUD.DNI

Si el campo es una variable estructurada, como en el ejemplo anterior que NOTAS es un array, se debe además especificar el subíndice para acceder a un elemento particular de dicho campo.

Por ejemplo, si quisiéramos mostrar el campo NOTAS:

```
FOR I:= 1 TO 6 DO
   WRITE (ESTUD.NOTAS[I]);
```

Como ya hemos dicho, las componentes de un registro pueden ser de cualquier tipo, o sea una componente de un registro puede ser también otro registro. Los registros que tienen este tipo de componentes se llaman registros jerárquicos.

Ejemplo:

```
TYPE  NOMB = ARRAY [1..20] OF CHAR;
        EMPLE = RECORD
```

```
                NOMBRE : NOMB;
                DIRECC : RECORD
                            CALLE : NOMB;
                            NUM : INTEGER;
                            PISO : INTEGER;
                            DPTO: CHAR
                          END;
                SUELDO : REAL
               END;
VAR
    EMPLEADO : EMPLE;
```

Un elemento particular de esta variable sería:

EMPLEADO.DIRECC.PISO

6-2. ARREGLOS DE REGISTROS:

Un arreglo de este tipo, es simplemente un arreglo cuyos elementos son registros.

Ejemplo: si tuviéramos el tipo de datos definido anteriormente, podríamos definir :

```
 TYPE  EMPRESA = ARRAY [1..200] OF EMPLE;
VAR   PERSONAL : EMPRESA;
```

Así:

```
WRITE (PERSONAL(4).SUELDO);
```

Nos mostraría el sueldo del empleado que está en la posición 4

EJERCICIO:

Un negocio de ventas al por mayor y al por menor, comercializa 100 productos distintos.-

El comerciante desea actualizar la lista de precios al público y de stock de los productos; para ello se ingresan, para cada uno de ellos, los siguientes datos:

- CODIGO (entero)

- DESCRIPCION (hasta 20 caracteres)

- CANTIDAD EN STOCK

- CANTIDAD MINIMA REQUERIDA EN STOCK

- PRECIO POR UNIDAD (para ventas al por menor)

- PRECIO POR UNIDAD PARA MAS DE 20 UNIDADES (para ventas al por mayor)

- PRECIO POR UNIDAD PARA MAS DE 50 UNIDADES (" " " " ")

Estos datos, que están ***ordenados en forma creciente*** por código de producto, se ingresarán por medio de un procedimiento.-

Luego se van ingresando los datos de los productos que tienen modificaciones (***no necesariamente los 100)***:

- CODIGO DEL ARTICULO (entero)

- CODIGO DE OPERACION ("C": compra; "V": venta)

- CANTIDAD (cantidad comprada o vendida)

- COEFICIENTE (coeficiente de ajuste del precio unitario)

Con la cantidad comprada o vendida se actualizará el stock.-

El coeficiente de ajuste se deberá multiplicar al precio unitario para obtener el nuevo precio (ventas por menor); y para obtener los precios por unidad para más de 20 o más de 50 unidades, se le aplicará ***al nuevo precio unitario obtenido*** un descuento del 10% y el 15% respectivamente.-

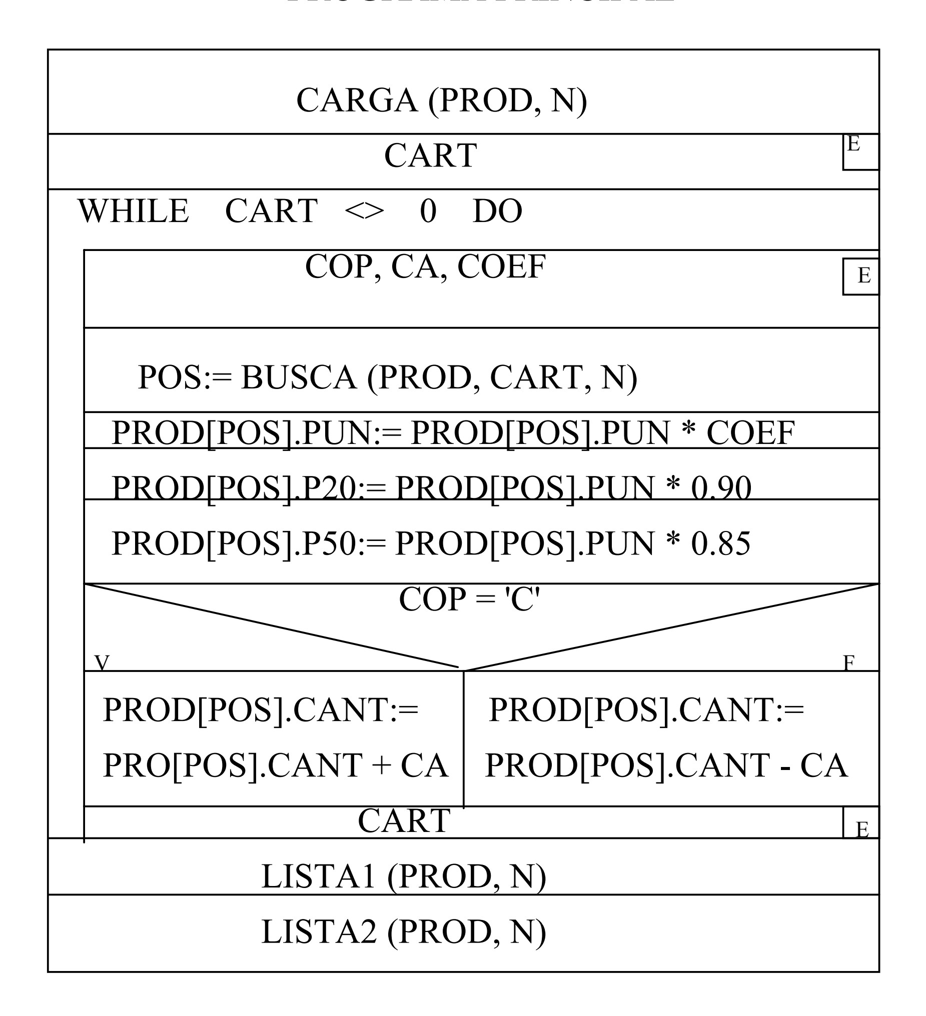
Se pide:

- Un listado actualizado de los 100 productos.-
- Un listado con los códigos y la descripción de los productos que tienen faltantes en stock con dicho faltante.-

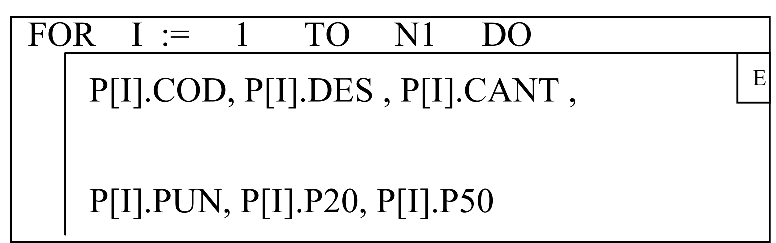
<u>NOTA:</u> La búsqueda del código a actualizar se hará por medio de una *FUNCION* que devuelva la posición en donde se encuentra dicho código (suponer que siempre se encontrará el mismo).-

a) <u>Diagrama de Chapin:</u>

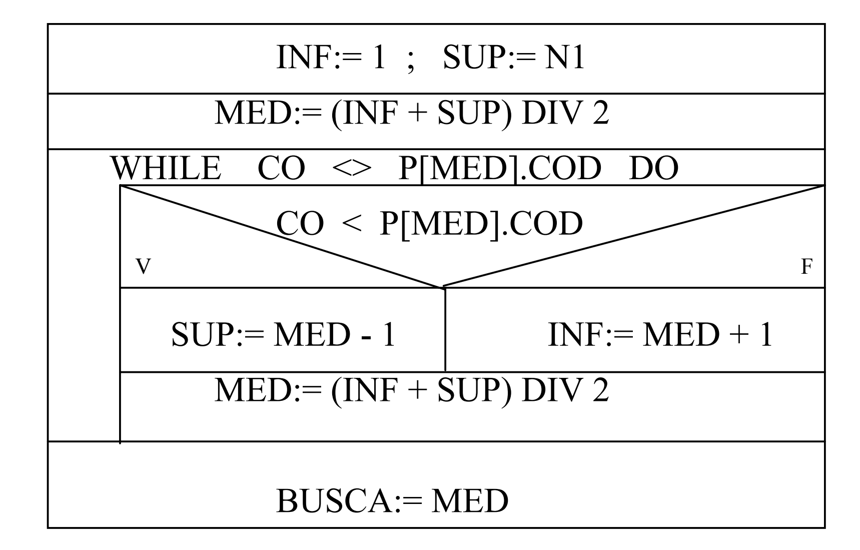
PROGRAMA PRINCIPAL



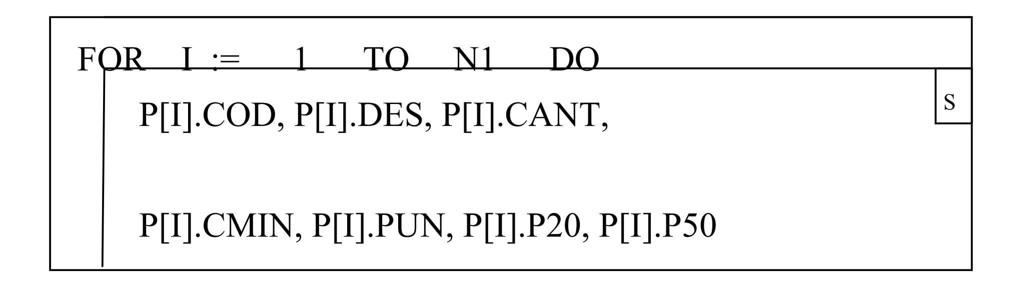
PROCEDURE CARGA (VAR P: PRODUCTO; N1: INTEGER);



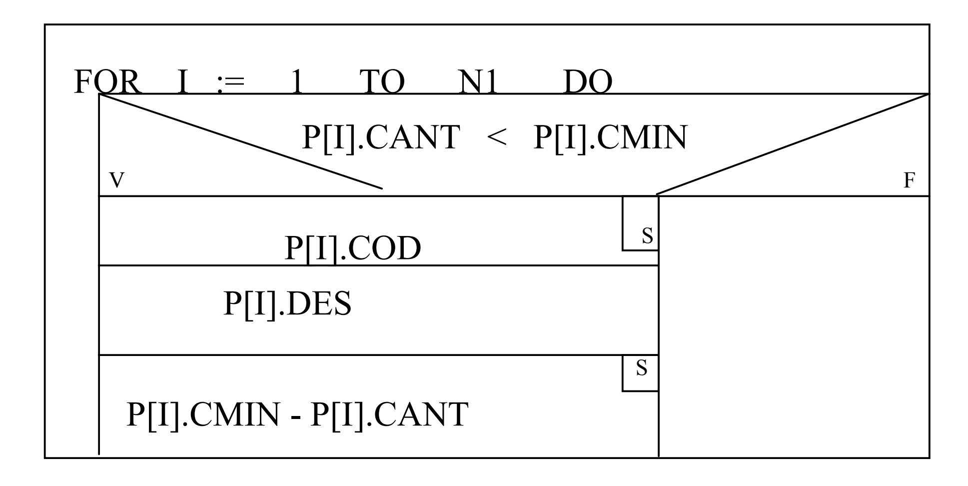
FUNCTION BUSCA (VAR P: PRODUCTO; CO, N1: INTEGER) : INTEGER



PROCEDURE LISTA1 (VAR P: PRODUCTO; N1: INTEGER)



PROCEDURE LISTA2 (VAR P: PRODUCTO; N1: INTEGER)



Programa PASCAL:

```
PROGRAM STOCK (input, output);
CONST
   N = 100;
TYPE
   ARTI = RECORD
            COD, CANT, CMIN: INTEGER;
            DES: STRING(20);
            PUN, P20, P50: REAL
        END;
   PRODUCTO = ARRAY [1..N] OF ARTI;
VAR
  PROD: PRODUCTO;
  CART, CA, POS: INTEGER;
  COP: CHAR;
  COEF: REAL;

PROCEDURE CARGA (VAR P: PRODUCTO; N1: INTEGER);
VAR
  I, C: INTEGER;
BEGIN
  FOR I := 1 TO N1 DO
  BEGIN
    WRITE ('INGRESE CODIGO');
    READLN (P[I].COD);
    WRITE ('INGRESE DESCRIPCION DEL ARTICULO');
    READLN (P[I].DES);
    WRITE ('INGRESE CANTIDAD');
    READLN (P[I].CANT);
    WRITE ('INGRESE CANT. MINIMA');
    READLN (P[I].CMIN);
```

```
    WRITE ('INGRESE PRECIO UNITARIO');
    READLN (P[I].PUN);
    WRITE ('INGRESE PRECIO UNIT. P/20 Y P/50');
    READLN (P[I].P20, P[I].P50)
  END
END;

FUNCTION BUSCA (VAR P: PRODUCTO; CO, N1: INTEGER) :INTEGER;
VAR
   INF, SUP, MED: INTEGER;
BEGIN
   INF:= 1 ; SUP:= N1 ;
   MED:= (INF + SUP) DIV 2;
   WHILE (CO <> P[MED].COD) DO
   BEGIN
      IF CO < P[MED].COD
         THEN SUP := MED - 1
         ELSE INF := MED + 1;
      MED := (INF + SUP) DIV 2
   END;
   BUSCA:= MED
END;

PROCEDURE LISTA1 (VAR P: PRODUCTO; N1: INTEGER);
VAR
  I, C : INTEGER;
BEGIN
   FOR I := 1 TO N1 DO
   BEGIN
     WRITE (P[I].COD);
     WRITE (P[I].DES);
```

```
    WRITE (P[I].CANT);
    WRITELN (P[I].PUN, P[I].P20, P[I].P50)
  END
END;

PROCEDURE LISTA2 (VAR P: PRODUCTO; N1: INTEGER);
VAR
  I, C: INTEGER;
BEGIN
  FOR I:= 1 TO N1 DO
    IF P[I].CANT < P[I].CMIN
      THEN
        BEGIN
          WRITE (P[I].COD);
          WRITE (P[I].DES[C]);
          WRITELN (P[I].CMIN - P[I].CANT)
        END
END;

BEGIN
  CARGA (PROD, N);
  WRITE ('INGRESE CODIGO ARTICULO');
  READLN (CART);
  WHILE CART <> 0 DO
  BEGIN
     WRITE ('INGRESE COD. OPERACION, CANT. Y COEF.');
     READLN (COP, CA, COEF);
     POS := BUSCA (PROD, CART, N);
     PROD[POS].PUN := PROD[POS].PUN * COEF;
     PROD[POS].P20 := PROD[POS].PUN * 0.9;
     PROD[POS].P50 := PROD[POS].PUN * 0.85;
```

```
    IF COP = 'C'
      THEN PROD[POS].CANT := PROD[POS].CANT + CA
      ELSE PROD[POS].CANT := PROD[POS].CANT - CA;
    WRITE ('INGRESE CODIGO ARTICULO');
    READLN (CART)
  END;
  LISTA1 (PROD, N);
  LISTA2 (PROD, N)
END.
```

UNIDAD Nº 7

ARCHIVOS

7-1- INTRODUCCION

Hasta ahora hemos visto diferentes estructuras de datos que son definidas en un algoritmo y ocupan memoria RAM, se utilizan en la ejecución y todos los valores contenidos en ellas se pierden, a lo sumo, cuando el algoritmo finaliza.

Para determinados problemas es necesario disponer de un tipo de estructuras de datos que permita guardar su información en soporte no volátil (disco, diskette, cinta, zip), y de esta forma preservarlas aunque el programa finalice. Estas estructuras de datos son conocidas además como estructuras de almacenamiento, y deben asociarse con un dispositivo de memoria auxiliar permanente donde archivar la información. Dichas estructuras se denominan archivos o ficheros.

7-2- ARCHIVOS

Un archivo es una estructura de datos que guarda en un dispositivo de almacenamiento secundario de una computadora, a una colección de elementos del mismo tipo.

Cuando se trabaja con un archivo en disco rígido se hace referencia a un conjunto de datos almacenados en memoria secundaria. El disco rígido puede almacenar un gran número de archivos.

Sin embargo, desde un programa la noción de archivo no es exactamente la misma. Un programa no sabe con exactitud el lugar donde residirán los datos, o sea , el lugar dentro del disco rígido donde estará el archivo físico; el programa envía y recibe información, desde y hacia el archivo. Para poder efectuar esto, el programa de aplicación se apoya en el sistema operativo de la computadora, que es el encargado de los detalles de almacenamiento, tales como lugar dentro del dispositivo físico, cantidad de espacio que va a utilizar, tipos de acceso permitido (si se puede leer, leer y escribir, si el acceso está vedado, etc.), usuarios que tienen acceso a los datos del

archivo, etc. El programa referencia directamente al archivo físico utilizando para ello un nombre lógico, que se denomina archivo lógico.

Antes de que el programa pueda operar sobre archivos, el sistema operativo debe recibir instrucciones para hacer un enlace entre el nombre lógico que utilizará el algoritmo y el archivo físico. Cuyo formato será, en líneas generales:

Asignar_correspondencia (nombre_físico, nombre_lógico)

En Pascal:

Assign (nombre_físico, nombre_lógico)

El nombre físico define exactamente el nombre con el que el sistema operativo encontrará al archivo dentro del almacenamiento secundario. En tanto, el nombre lógico se corresponde con una variable definida en el algoritmo. Dicha variable debe ser de tipo archivo.

La declaración de un archivo en Pascal se hará de la siguiente forma:

Var archivo: file of tipo_de_datos;

ó:

Type archivo = file of tipo_de_datos;
Var archi: archivo;

7-3- BUFFERS

Se denomina buffer a una memoria intermedia entre un archivo y un programa, donde los datos residen provisoriamente hasta ser almacenados definitivamente en memoria secundaria o donde los datos residen una vez recuperados de dicha memoria secundaria. Los buffers ocupan una zona de la memoria RAM de la computadora.

Manejar buffers implica trabajar con grupos de datos en memoria RAM para que el número de accesos al almacenamiento secundario se reduzca. Básicamente las operaciones de lectura y escritura no se realizan directamente sobre la memoria secundaria. Si esto fuera así, se necesitaría, ante cada uno de estas operaciones una determinada cantidad de milisegundos para realizarla. Por lo tanto, estas operaciones que se definen en los algoritmos, interactúan con un buffer, el cual al encontrarse en memoria RAM agiliza el proceso.

El sistema operativo de la computadora es el encargado de manipular los buffers. Cuando un programa realiza una operación de lectura y en el buffer no hay información para satisfacerla, se lee de memoria secundaria los datos para completar nuevamente dicho buffer y así poder satisfacer el requerimiento. Algo similar ocurre con la escritura, cuando se intenta escribir en un buffer y el mismo no tiene capacidad, la información contenida es bajada o guerdada en memoria secundaria, una vez vacío el buffer puede tomar los datos definidos en la orden de escritura.

7-4- OPERACIONES BASICAS SOBRE ARCHIVOS

7-4-1- APERTURA Y CREACIÓN

- Para abrir un archivo existente en memoria secundaria como lectura y escritura:

Reset (nombre_lógico);

- Para crear un archivo, o sea para abrir un archivo de sólo escritura:

Rewrite (nombre_lógico);

Donde nombre_lógico es la variable de tipo archivo sobre la que se realizó la asignación correspondiente.

Téngase en cuenta que si se realiza la apertura con la instrucción Rewrite y el archivo contiene información, la misma se perderá completamente.

7-4-2- CIERRE

Para efectuar el cierre explícito de un archivo y colocar la marca de fin de archivo:

Close (nombre_lógico);

7-4-3- LECTURA Y ESCRITURA

Para leer datos de un archivo:

Read (nombre_lógico, variable);

Para escribir datos en un archivo:

Write (nombre_lógico, variable);

Donde variable es una variable cuyo tipo de dato debe corresponderse con la definición del archivo.

Cada una de estas instrucciones opera sobre la posición actual del archivo, y luego avanza a la posición siguiente.

7-5- OTRAS OPERACIONES SOBRE ARCHIVOS

7-5-1- FIN DE ARCHIVO

Para recorrer un archivo desde el primer elemento hasta el final, es necesario contar con operación que detecte el fin de dicho archivo:

Eof (nombre_lógico);

Esta función devuelve un valor booleano que será True si la posición corriente dentro del archivo referencia a la marca de fin, y False, en caso contrario.

7-5-2- NÚMERO DE ELEMENTOS DEL ARCHIVO

Si necesitamos saber la cantidad de elementos o registros que posee un archivo:

Filesize (nombre_lógico);

Esta función devuelve un número entero, que corresponde a la cantidad de registros del archivo.

7-5-3- POSICIÓN ACTUAL

A veces necesitamos operar en otros procesos con la posición actual del archivo, para lo cual debemos poder determinarla:

Filepos (nombre_lógico);

Esta función devuelve un número entero que corresponde a la posición actual del apuntador del archivo.

7-5-4- POSICIONARSE EN UN ELEMENTO

Para modificar el contenido de un elemento particular de un archivo necesitamos posicionarnos en esa dirección:

Seek (nombre_lógico, posición);

Esta función permite llegar a un elemento particular del archivo. Donde posición es un número entero menor a la cantidad de elementos del archivo.

7-6- ORGANIZACIÓN Y ACCESO A UN ARCHIVO

Según el modo en que se organizan los registros dentro de un archivo, se consideran dos tipos de acceso:

- **Acceso secuencial**
- **Acceso directo**

El acceso secuencial permite acceder a los elementos o registros uno tras otro y en el orden físico en que están guardados. El acceso directo, en cambio, permite obtener un registro determinado sin necesidad de haber accedido a los anteriores.

De acuerdo a su organización, un archivo puede clasificarse como:

- **Directo**
- **Secuencial**
- **Secuencial Indizado**

Esta organización define la manera en que los registros se distribuyen sobre el almacenamiento secundario.

Un archivo secuencial consiste de un conjunto de registros almacenados consecutivamente de manera que para acceder al registro n-ésimo se debe, previamente, acceder a los n-1 registros anteriores. Los registros se graban en forma consecutiva, a medida que se ingresan, y se recuperan en el mismo orden.

Un archivo directo consiste de un conjunto de registros donde el ordenamiento físico no necesariamente corresponde con el ordenamiento lógico. Los registros se recuperan accediendo por su posición dentro del archivo. Por lo tanto es posible acceder al n-ésimo lugar sin haber accedido a los n-1 registros anteriores. Esta organización presenta la ventaja que se puede obtener cualquier elemento del archivo en cualquier orden, siendo muy eficientes en cuanto a tiempo de acceso necesario para recuperar la información; pero presenta el inconveniente de tener que determinar la posición donde se encuentra cada elemento.

Un archivo secuencial indizado utiliza estructuras de datos auxiliares para permitir un acceso pseudo directo a los registros del archivo. Un ejemplo de esta organización es la guía telefónica, en la que se puede acceder por letra, y dentro de cada página existe una indicación de apellido de comienzo y apellido de fin dentro de la hoja; de esta forma se puede acotar el espacio de búsqueda de un determinado teléfono dentro de la guía, haciendo referencia mucho más rápidamente a la hoja donde se encuentra el dato. Los archivos organizados con esta técnica tienen la ventaja de tener un acceso mucho más rápido que los secuenciales, pero necesitan más espacio para mantener las estructuras de los índices. Estas estructuras se denominan directorios del archivo.

7-7- ARCHIVOS DE TEXTO

Pascal estándar define dos archivos de texto (TEXT) por defecto que son los archivos **INPUT** y **OUTPUT**, que se corresponden con los periféricos estándar de entrada y salida en la instalación, normalmente el teclado y la pantalla o impresora.

Los procedimientos **READ** y **WRITE** cuando omiten el archivo empleado, se sobreentiende que se emplea estos dos archivos por defecto, por ello es obligatorio su utilización como parámetros en el encabezamiento del programa, no así en Turbo Pascal, donde se pueden omitir.

Los archivos de texto se dividen en líneas formadas por conjuntos de caracteres, separadas unas de otras por caracteres de control especiales. En el código ASCII, la marca separadora de líneas está constituida por la combinación de caracteres *CR/LF* (retorno de carro/avance de línea).

En este tipo de archivos también se utiliza la variable booleana Eoln (f) para indicar si el buffer se encuentra o no sobre la marca separadora de líneas. En caso afirmativo, toma el valor true. En algunas implementaciones de Pascal, el contenido del buffer cuando se encuentra en la marca de fin de línea es un carácter en blanco (#32), por lo que debe tenerse esto en cuenta, por ejemplo, si hacemos una estadística de caracteres de un archivo de texto.

Las funciones **Eoln** y **Eof** se refieren al fichero **Input** a menos que se especifique un fichero o archivo diferente, en ese caso sería **Eoln(nombre_lógico)** y **Eof(nombre_lógico)**.

7-8- EJERCICIO

Escribir un procedimiento que actualice los salarios de los empleados de una empresa, aumentándolos un 10%, considerando que el archivo ya existe, que la asignación nombre físico, nombre lógico está realizada en el programa principal y que el archivo no está abierto.

El archivo debe quedar actualizado y cerrado.

Supongamos que en el programa principal está hecha esta declaración:

```
Type   registro = Record
              Nombre: string(20);
              Dirección: string (20);
```

```
                Salario: real
              End;
             Empleados = file of registro;
.
.
.
Procedure Actualizar (var Emp: Empleados);     {se recibe el archivo como parámetro
                                                 por referencia}
Var    E: registro;
Begin
     Reset (Emp);                    {el archivo contiene datos, se abre de E/S}
     While not eof (Emp) do          {se evalúa si no se llegó a la marca de fin de  archivo}
       Begin
          Read (Emp, E);        {se obtiene el elemento del archivo}
          E.salario := E.salario * 1.1;      {se incrementa el salario}
          Seek (Emp, filepos (Emp) – 1);  {luego de la lectura la posición corriente del
                                            archivo avanza una posición, para hacer la
                                          escritura se debe retroceder en uno dicha posición}
          Write (Emp, E);
       End;
     Close (Emp)
End;
```